Brady

INCLUDES A DISK

George Sutty and Steve Blair

Bestselling Authors of *Advanced Programmer's Guide to the EGA/VGA*

# ADVANCED PROGRAMMER'S GUIDE TO SuperVGAs

## The Advanced Programmer's Graphic Library Volume II

# Advanced Programmer's Guide to SuperVGAs

George Sutty
Steve Blair

Brady

New York London Toronto Sydney Tokyo Singapore

BRADY

Simon & Schuster, Inc.
15 Columbus Circle
New York, NY 10023

DISTRIBUTED BY PRENTICE HALL TRADE

Manufactured in the United States of America

10 9 8 7 6 5 4 3 2 1

**Library of Congress Cataloging-in-Publication Data**

Sutty, George J.
Advanced programmer's guide to SuperVGAs [computer file].
1 computer disk ; 5¼ in. + 1 v.
System requirements: IBM PC, XT, AT, PS/2, or compatibles; 640K; MS-DOS 3.0 or higher; one 1.2M disk drive; monochrome monitor; SuperVGA board.
Title from t.p. of book.
Not copy-protected.
Audience: Adult reference/general.
Summary: Offers a graphics programming utility with drawing routines for 256, sixteen, and four color graphics. The text is designed to enable users to utilize fully the many advanced features of the SuperVGAs.
1. Computer graphics—Software. I. Blair, Steve. II. Title.
T385.S84 1990 004.7 90-69814
ISBN 0-13-010455-8

For information about our audio products, write us at:
Newbridge Book Clubs, 3000 Cindel Drive, Delran, NJ 08370

# Dedication

To our families for their support and encouragement.

# Acknowledgments

We would like to thank all of the VGA manufacturers who were gracious enough to supply us with technical material and sample products, especially those who participated in the review of the manuscript. This book would not have been possible without their support.

We would particularly like to thank Mr. Gary Lorensen, whose expert and thorough review of the manuscript (in several different versions) contributed greatly to the quality of the finished result.

Special thanks are also due Susan Hunt and the rest of the team at Brady Books for their patience, encouragement, and assistance with the manuscript.

# Contents

# Part I

## *A VGA Review*

# Introduction

Since the introduction of the Apple Macintosh computer system in 1984, which gained wide acceptance as an easy-to-learn and easy-to-use tool to increase productivity, nearly all new user interfaces for computers have been graphics based. Such popular interfaces as Microsoft Windows, IBM Presentation Manager, GEM by Digital Research, NewWave by Hewlett-Packard, X-Windows, NextStep, NeWS and Open Look are all graphics based. Graphical interfaces offer many advantages over text-based interfaces, and are especially useful in personal computers where a user-friendly, interactive interface can have big payoffs in productivity and ease of use.

Increasingly, today's most popular software applications (Aldus Pagemaker, Microsoft Excel, Ventura Publisher, and others) rely on Graphical User Interfaces (GUIs) to interact efficiently with the user. It appears that GUIs are destined to become the standard for personal computers. This also means that graphics programming is becoming less of a programming specialty and more the standard method of implementing applications.

The increasing popularity of GUIs has been made possible by rapid advances in graphics technology. In just a few years, personal computer graphics has progressed from nonexistent to the breathtaking quality of the VGA and SuperVGA, which are quickly becoming the most prevalent graphics adapters for IBM-compatible computers. The superior features and affordable price of the VGA guarantee it an increasing share of the IBM-compatible video market. International Data Corporation predicted that in 1989 1.7 million EGA boards will be shipped, compared with 1.4 million VGA boards. By 1993, however, IDC predicts that VGA (including SuperVGA) will be the most prevalent display adapter in use, with more than 11 million units installed.

# Why We Wrote this Book

The IBM VGA has been well documented in several texts. We are partial to our previous text, the *Advanced Programmer's Guide to EGA/VGA*, as a convenient reference on the subject. Most of the VGA boards on the market, however, offer significant features (such as high resolution display modes) that are not available on the IBM product and are not covered in any of the texts on VGA. The added benefits of SuperVGA boards are not at all trivial; SuperVGAs can now be used in applications for which the original IBM VGA is ill-suited.

This book contains the information you need to know to write software that takes advantage of the benefits of SuperVGAs. Its sole purpose is to provide a complete and comprehensive tutorial on SuperVGAs. It tells you, among other things, what SuperVGA features are available, how to select a SuperVGA that suits your needs, how to utilize advanced SuperVGA features in your software, and how to write graphics drawing routines that will work in the high resolution modes of your SuperVGA. It

explains the differences between SuperVGAs, as well as their similarities. It describes what displays are available and offers suggestions on how to select one.

In short, this book is designed to enable you to utilize fully the many advanced features of the SuperVGAs.

# The History of VGA

While IBM has repeatedly demonstrated its ability to use a combination of engineering and marketing to set important standards for personal computing, many competitors have shown their ability to improve on IBM's standards by introducing products that are IBM compatible but with additional features and enhancements which are not included in the IBM product.

Introduced in 1982 with the IBM PC, the Monochrome Display Adapter (MDA) was IBM's first video product for personal computers. The MDA is a text mode display adapter, and offers no graphics or color capabilities. Shortly thereafter, Hercules Computer Technology Inc. introduced the Hercules Monochrome Graphics Adapter, which is MDA compatible but offers a graphics mode as well. Hercules established the first independent video standard for IBM computers.

MDA was followed by the Color Graphics Adapter (CGA), which offers relatively crude color graphics modes. CGA text is actually less readable than that of the MDA, and CGA gained acceptance only among those who had a strong desire for its color graphics capabilities. Several manufacturers introduced enhanced CGA products, but due to a lack of standardization their product enhancements were largely ignored.

In 1985 IBM introduced the Enhanced Graphics Adapter (EGA). The EGA offers color graphics modes which are superior to those of the CGA, and includes some compatibility with MDA and CGA as well. With EGA, IBM began a new product trend; it was the first PC video adapter to be based on proprietary (VLSI) technology. This made the task of building a compatible product much more difficult for IBM's competitors. Around the same time the Professional Graphics Controller (PGC) became available. Although this board provided 256 colors in 640x480, the high cost of the adapter and even higher cost of display needed caused this board to capture only a very small segment of the PC market.

Despite its advantages, the cost of the IBM EGA kept it from becoming widely accepted until several chip manufacturers (Chips and Technologies, Paradise, Tseng Labs, ATI, and others) engineered the VLSI devices required to clone the IBM product, adding enhancements as they did so. IBM quickly became an insignificant force in the EGA market.

The IBM Video Graphics Array (VGA) was introduced by IBM in April 1987 as the standard display interface for the PS/2 line of personal computers, except models 25 and 30 which are equipped with the Multi-Color Graphics Array (MCGA). VGA is similar to EGA, but is the first IBM display adapter to use an analog display interface (previ-

ous adapters used digital display interfaces). This gives the VGA much greater color capabilities than the EGA. Before VGA, 256-color capability in the IBM PC-compatible arena was available only with high-end graphics products. The MCGA has not become popular and is found only in low-end PS/2 products from IBM.

While the PS/2 and its Micro Channel have managed to capture only a small percentage of the personal computer market, the VGA quickly gained widespread market acceptance as the display standard of the future. EGA chip manufacturers soon repeated their success by cloning the IBM VGA. Their enhanced VGA products, which offer higher resolutions and more colors than the IBM product, have been nicknamed the SuperVGAs.

By October 1987, STB Systems and Sigma Designs were shipping register-compatible VGA boards, and many other vendors were announcing similar products. By late 1987, enhanced VGA products were appearing with 800 by 600 resolution, the greater detail quickly becoming popular for desktop publishing and Computer-Aided Design (CAD). By late 1988 many vendors were offering 256-color modes at resolutions of 640 by 400, 640 by 480, and even 800 by 600.

Unfortunately, the displays available at the time were less than ideal at this resolution, exhibiting noticeable flicker and data degradation. As displays with higher bandwidth and scanning rates became available, VGA board vendors added support for 1024 by 768 resolution.

VGA chip vendors now appear to be the driving force behind the personal computer graphics industry. By designing newer and better VGA chips, they have defined the features and enhancements that are found in the newest SuperVGA products.

The fly in all this graphics ointment is that there is a lack of standardization among SuperVGA vendors as to how enhancements have been added. This has made it difficult for software developers to utilize the enhanced features. SuperVGA board vendors have partially alleviated this problem by supplying drivers with their products to utilize the enhanced features with popular software packages such as Microsoft Windows, GEM, Autocad, Lotus 1-2-3, and others.

Recognizing the virtues of standardization, a number of SuperVGA vendors have now banded together to form the Video Electronics Standards Association (VESA), a standards committee committed to, among other things, the development of an expanded VGA standard. VESA has also assumed the task of standardizing the interface for the high resolution displays that are needed to take advantage of SuperVGA technology. It will be a rare accomplishment if this industry organization succeeds in setting a personal computing standard without the assistance of IBM.

## How the Book is Organized

The *Advanced Programmer's Guide to SuperVGAs* is intended to satisfy the growing need for detailed information regarding enhanced VGA products. It explains how to

write software that can utilize the advanced modes and features of the SuperVGAs. It includes useful graphics algorithms tailored for the SuperVGA. The book is intended to be used both as a tutorial text and as a reference source where answers to VGA related questions can be found. It is a companion text to the *Advanced Programmer's Guide to EGA/VGA*, which offers the reader a complete tutorial regarding the IBM EGA and VGA display adapters and compatible products. An understanding of the information in that text is prerequisite for much of the information that is presented here. This text will concentrate on SuperVGA features that are not found on the standard VGA. It is assumed that the reader has some familiarity with the BIOS, register set, and display memory of the standard IBM VGA, as well as the Intel 8086/80286 assembly language instruction set.

The book consists of two major sections:

Part I contains an overview of the VGA architecture. Basic principles of the VGA, its register set, display memory, and ROM BIOS are discussed. This chapter is not intended to be a substitute for the in-depth description of the VGA given in our previous text, the *Advanced Programmer's Guide to EGA/VGA*. It is included here as a summary and reference source.

Part II covers basic principles that are common among SuperVGAs. While SuperVGA implementations differ from vendor to vendor, basic design requirements are the same for all. These include issues such as how to address the much larger display memory that is required, what new display resolutions to support, what new BIOS support to add, and what software drivers to supply. It includes a discussion of the incompatibilities between some of the popular SuperVGA products. Programming examples illustrate how to manipulate registers and read back VGA status information. Only products that are capable or 256 colors at resolutions of 640x400 or better are considered here to be SuperVGAs.

Part II of the book provides detailed descriptions of some of the most popular SuperVGA products (one per chapter). The characteristics of a particular VGA product are for the most part determined by the manufacturer of the VLSI integrated circuit that is the heart of any VGA design. VGA products from several different board vendors may be very compatible if they are based on the same VLSI device. This section describes VGA products based on commonly found VLSI devices. The result is that most VGA products currently on the market will conform closely to one of these descriptions.

The final chapter of the book is dedicated to VGA-compatible displays. A large number of high quality, high resolution color displays are now available, each with its own advantages and disadvantages. Included is a summary of display terminology and information needed to evaluate different displays. Resolution, bandwidth, pixel size, tube size, aspect ratio and sync timing are just some of the factors that affect the quality and flexibility of a display.

The appendices of the book contain tables that summarize VGA information for quick reference, along with a glossary of terms.

# 1

# *Standard VGA Display Modes*

# Introduction

IBM introduced the Video Graphics Array (VGA) display adapter in 1985 as the standard display adapter for their PS/2 computer systems. While acceptance of the PS/2 and its Micro Channel has been mixed, the VGA has been widely embraced as the graphics adapter technology of today and tomorrow.

Unlike most earlier display adapters, which drive displays with a TTL digital interface, VGA adapters drive analog displays. This makes it possible for the VGA to display many more colors than other display adapters (including EGA). It also means, however, that the VGA is incompatible with many existing displays.

As the VGA and its analog monitors become more widespread, compatibility will become much less of an issue. Many programs written for other color display adapters cannot operate with monochrome displays, and vice versa. This is not a problem with the VGA; if a monochrome display is attached, color information is automatically converted to shades of gray. Monochrome information can also be shown on a color screen.

Both color and monochrome VGA-compatible displays are available from a wide variety of sources. Part 3 of this text discusses popular VGA displays.

Like its predecessors, the VGA is a nonintelligent display device; it has no on-board drawing or processing capability. The system processor must perform all drawing functions by writing directly to display memory. Essentially, writing one bit into display memory is equivalent to lighting one pixel on the display screen. Most of the circuitry of the VGA is dedicated to the task of transferring the data in display memory onto the display screen. This process, called display refresh, must be performed between 50 and 70 times each second.

In color display systems, the number of colors that can be displayed on the screen at one time is governed by the number of bits of display memory that are dedicated to color information for each pixel. If n bits per pixel are used, $2^n$ colors can be generated. VGA uses from one to eight bits per pixel, permitting up to 256 ($2^8$) colors to be displayed on the screen at the same time. In other words, the VGA is capable of 256 simultaneous colors.

In order to standardize the video interface for applications software, IBM defined a set of standard display modes for the VGA. SuperVGA vendors have added to the list of standard modes by creating new high resolution display modes. These modes do not represent all configurations in which the display adapter can operate, but there are few good reasons to stray from the defined standard modes. Many of the standard VGA display modes have been carried forward from the MDA, CGA, and EGA display adapters.

Table 1.1 lists the display modes that are available for the standard VGA.

Table 1-1. Standard IBM VGA video modes

| Mode | Type | Resolution | Colors |
|---|---|---|---|
| 0,1 | Text | 40 columns x 25 rows (320x200, 8x8 char cell) | 16 |
| 0* | Text | 40 columns x 25 rows (320x350, 8x14 char cell) | 16 |
| 0+ | Text | 40 columns x 25 rows (320x400, 9x16 char cell) | 16 |
| 2,3 | Text | 80 columns x 25 rows (640x200, 8x8 char cell) | 16 |
| 2* | Text | 80 columns x 25 rows (640x350, 8x14 char cell) | 16 |
| 2+, 3+ | Text | 80 columns x 25 rows (640x400, 9x16 char cell) | 16 |
| 4,5 | Graphics | 320 horizontal x 200 vertical | 4 |
| 6 | Graphics | 640 horizontal x 200 vertical | 2 |
| 7 | Text | 80 columns x 25 rows (720x350, 8x14 char cell) | Mono |
| 7+ | Text | 80 columns x 25 rows (720x400, 9x16 char cell) | Mono |
| D | Graphics | 320 horizontal x 200 vertical | 16 |
| E | Graphics | 640 horizontal x 200 vertical | 16 |
| F | Graphics | 640 horizontal x 350 vertical | Mono |
| 10h | Graphics | 640 horizontal x 250 vertical | 16 |
| 11h | Graphics | 640 horizontal x 480 vertical | 2 |
| 12h | Graphics | 640 horizontal x 480 vertical | 16 |
| 13h | Graphics | 320 horizontal x 200 vertical | 256 |

Since the introduction of the IBM Color Graphics Adapter (CGA), all IBM display adapters have included 40 column text modes. These modes were created to allow text to be displayed on home television sets, which have much poorer resolution than computer displays and cannot display 80 columns of text. Other than a small number of computer games which have been written using 40-column text, these modes are not commonly used.

Special adapter circuitry is required to connect an IBM compatible computer to a television set (unless the TV set can accept composite video input).

# Standard VGA Display Modes

## Modes 0 and 1 (40-column Color Text)

On the VGA there is no functional difference between mode 0 and mode 1. These two modes were brought forward from the CGA video adapter and the distinction between them disappeared with the CGA Composite Video output jack. Modes 0 and 1 display color text at a resolution of 40-character columns by 25-character rows.

CGA compatibility is not complete, and not all CGA software will run properly in these modes. In general, software which makes use of BIOS video services and avoids any direct access to I/O registers on the video adapter will usually run without prob-

lems. Direct processor access to display memory does not cause compatibility problems.

## Modes 2 and 3 (80-column Color Text)

Modes 2 and 3 are the 80-column counterparts to the 40-column modes 0 and 1. On the VGA, there is no functional difference between mode 2 and mode 3. As with modes 0 and 1, these two modes were brought forward from the CGA video adapter and the distinction between them disappeared with the CGA Composite Video output jack. Modes 2 and 3 display color text at a resolution of 80-character columns by 25-character rows.

### Double Scanning

When operating in CGA-compatible graphics modes, the VGA display adapter uses a technique known as DOUBLE SCANNING to display the low resolution (200 scan line) CGA display on the high resolution (400 scan line) VGA display. Each of the 200 horizontal scan lines is displayed twice, increasing the vertical screen resolution from 200 scan lines to 400 scan lines. This improves the quality of the display, and helps compensate for the different aspect ratio of the VGA display. Double Scanning is used for modes 4,5,6,D, and E.

## Modes 4 and 5 (Four-color 320x200 Graphics)

Modes 4 and 5 are very popular CGA graphics modes which were also carried forward to EGA and VGA. The distinction between these modes disappeared with the CGA Composite Video output jack. Display resolution is 320 pixels horizontally by 200 pixels vertically. The VGA uses double scanning to increase this to 400 lines vertically.

Four-color pixel data is stored in a packed pixel format with two bits per pixel. Details are given in the section "Display Memory in Graphics Modes."

As with all standard CGA modes, compatibility is not complete. Software which writes directly to I/O registers of the CGA may not function properly on VGA. Software which makes use of BIOS calls to configure the registers will usually operate properly.

### CGA Graphics Modes

These modes present an unusual set of challenges for the graphics programmer because the display memory is not linearly mapped. A computation is required to translate from a pixel location on screen to a location in display memory. For an explanation of the CGA graphics memory map, see the section "Display Memory in Graphics Modes".

## Mode 6 (Two-color 640x200 Graphics)

Mode 6 is the highest resolution graphics mode of the CGA, carried forward to VGA. A screen resolution of 640 pixels horizontally by 200 lines vertically is supported, but only in two colors. The VGA uses double scanning to increase this to 400 lines vertically.

As with all standard CGA modes, compatibility is not complete. Software which writes directly to I/O registers of the CGA may not function properly on EGA. Software which makes use of BIOS calls to configure the registers will usually operate properly.

As explained for modes 4 and 5, the display memory is not linearly mapped. A computation is required to translate from a pixel position on the screen to an address in display memory. Details are given in the section "Display Memory in Graphics Modes."

## Mode 7 (Monochrome Text)

In mode 7 the VGA is partially software compatible with the Monochrome Display Adapter (MDA). The display is formatted as 80 character columns by 25 character rows.

In monochrome text mode, character attributes do not control character color but represent other display characteristics. Monochrome text attributes include character blink, intensify, underline, and reverse video. Monochrome text attributes are described in detail later in this chapter.

## Mode D (Sixteen-color 320x200 Graphics)

Unlike previously described modes, this mode is not a backward compatibility mode for CGA or MDA; it exists for EGA and VGA only. It is loosely patterned after mode 4 (CGA 4-color graphics), but offers more colors. The limited resolution of mode D (320 horizontal pixels by 200 vertical lines) makes it undesirable for new software applications, yet it is not software compatible with any older applications. The result is that mode D is rarely used. The VGA uses double scanning to increase the screen size to 400 lines vertically.

Mode D does not suffer from the nonlinear memory mapping that CGA compatible graphics modes do, and translating from a pixel position on the screen to a location in display memory is relatively straightforward. The memory map for mode D is described in the section "Display Memory in Graphics Modes".

## Mode E (Sixteen-color 640x200 Graphics)

Like mode D, mode E exists for the EGA and VGA only. It is loosely patterned after CGA mode 6 (two-color graphics), but offers more colors. Its limited resolution (640 pixels horizontally by 200 lines vertically) makes it unpopular for new software devel-

opment, and it is not compatible with any older existing software. The result is that mode E is rarely used. The VGA uses double scanning to increase the screen size to 400 lines vertically.

Mode E does not suffer from the nonlinear memory mapping that CGA compatible graphics modes do, and translating from a pixel position on the screen to a location in display memory is relatively simple. Details are given in the section "Display Memory in Graphics Modes."

## Mode F (Monochrome 640x350 Graphics)

Graphics mode F is unique to the EGA and VGA. Resolution in mode F is 640 pixels horizontally by 350 lines vertically, which is less than the 720 horizontal by 348 vertical resolution of the Hercules monochrome graphics adapter.

Mode F does not suffer from the nonlinear display memory address mapping of the Hercules adapter. The display memory is linearly mapped.

Two "color" planes of display memory are used, giving each monochrome pixel four attributes. These attributes are:

00 - black
01 - white
10 - blinking
11 - intensified

The memory planes can be enabled and disabled independently by writing to the plane enable register, index 2 in the Sequencer.

## Mode 10 (Sixteen-color 640x350 Graphics)

Mode 10, which is unique to the EGA and VGA, is the most popular mode for new color graphics applications. It supports a resolution of 640 horizontal pixels by 350 vertical pixels. Four color planes are used, yielding up to 16 simultaneous colors. Color planes are enabled and disabled by writing to the plane enable register in the Sequencer.

## Mode 11 (Two-color 640x480 Graphics)

Mode 11 supports the IBM VGA at its highest standard resolution (640 pixels horizontally by 480 lines vertically), but supports only two simultaneous colors. This mode can be used to display 30 rows of 80 column text.

## Mode 12 (Sixteen-color 640x480 Graphics)

Mode 12 supports the VGA at its highest resolution (640 pixels horizontally by 480 lines vertically), with 16 simultaneous colors. This is a popular mode for new color graphics applications. Four color planes are used, yielding up to 16 simultaneous colors. Color planes are enabled and disabled by writing to the plane enable register, index 2, in the Sequencer.

## Mode 13 (256-color 320x200 Graphics)

This mode, which is unique to the VGA, is the only 256-color mode of the standard VGA. Resolution is limited to only 320 pixels horizontally by 200 lines vertically, which is double scanned to increase the vertical height to 400 lines.

# 2

# *Architecture of the VGA*

# Introduction

With the exception of the video output DAC (Digital to Analog Converters), the architecture of the VGA closely resembles that of the EGA. The VGA includes a few additional registers, and lacks the light pen support of EGA. Unlike many of the EGA registers, most VGA registers include read-back capability; the lack of read-back ability was such a drawback in the original EGA that it was added later by most EGA chip manufacturers.

## Packed PIxels vs. Color Planes

Two common techniques for storing color information are the packed pixel method and the color plane method. The original EGA is color plane oriented, except for the CGA-compatible modes, modes 4 through 5, which use packed pixels. VGA has one added mode, the 256-color packed pixel mode.

With packed pixels, all color information for a pixel is packed into one word of memory data. For a system with few colors, this packed pixel may require only part of one byte of memory; for very elaborate systems, a packed pixel might be several bytes long. Using 8 bits per pixel, a packed pixel looks as shown in Figure 2-1.

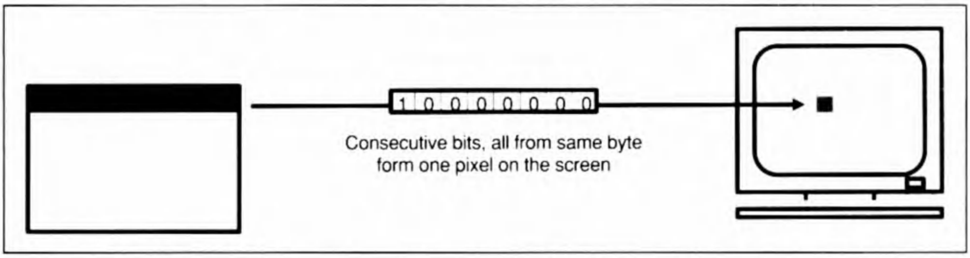


**Figure 2-1. Packed pixels**

With the color plane approach, the display memory is separated into several independent planes of memory, with each plane dedicated to controlling one color component (such as red, green, or blue). Each pixel of the display occupies one bit position in each plane. This approach is shown in Figure 2-2.

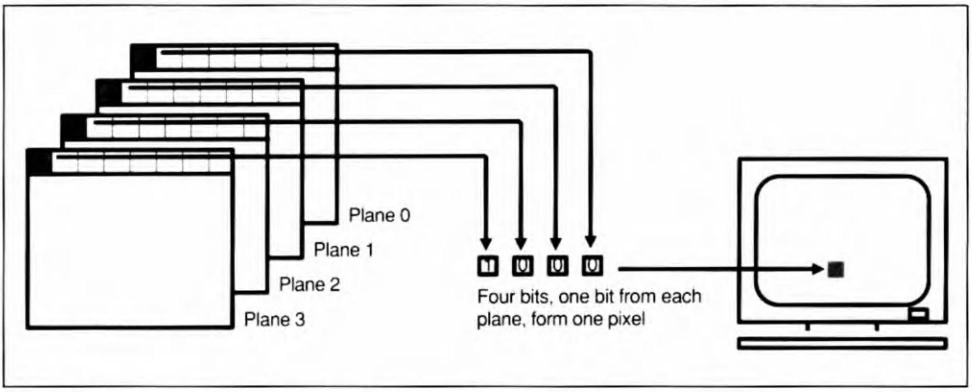


Figure 2-2. Planar pixels

## Text Modes vs. Graphics Modes

Two basic types of operating modes exist for the VGA: text mode and graphics mode. In graphics modes (which IBM frequently refers to as **All Points Addressable** modes), a set of bits in display memory represents a single pixel on the display screen. In text modes, however, a single byte ASCII character code placed in display memory causes an entire text character to be displayed on the screen. Text modes require much less display memory and place less burden on the system processor, but they are very limited in that only text and crude block graphics objects can displayed. Figure 2-3 illustrates the basic operation of a text mode, and Figure 2-4 shows the operation of a graphics mode.

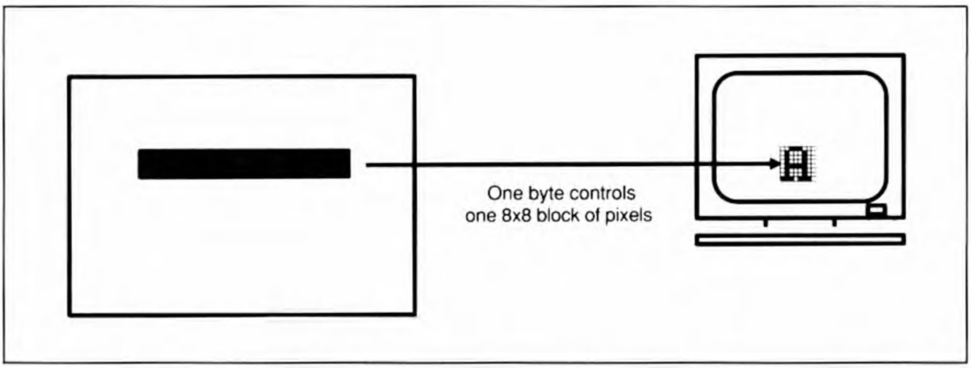


Figure 2-3. Text mode operation

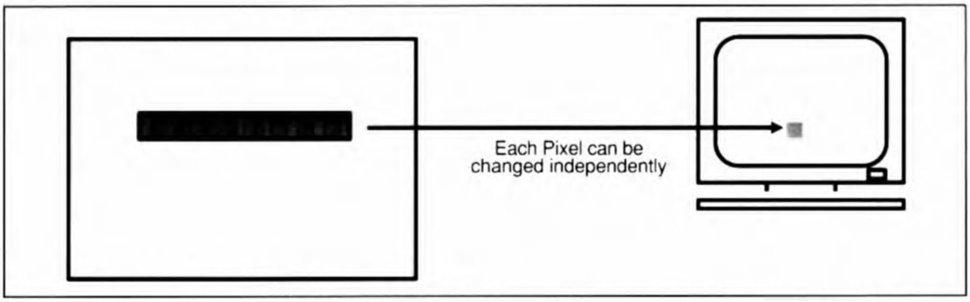


Figure 2-4. Graphics mode operation

## Functional Blocks

Figure 2-5 illustrates the basic architecture of the VGA, which consists of six major functional blocks:

- The **Display Memory** is a bank of 256 K (or more) of dynamic random access memory (DRAM or VRAM), divided into four planes, which holds the screen display data.
- The **Graphics Controller** resides in the data path between the processor and display memory. It can be programmed to perform logical functions (such as AND, OR, XOR, or ROTATE) on data being written to display memory. These logical functions can provide a hardware assist to simplify drawing operations.
- The **CRT Controller** generates timing signals (such as syncing and blanking) to control the operation of the CRT display and display refresh timing.
- The **Data Serializer** captures display information which is taken from display memory one or more bytes at a time, and converts it to a serial bit stream to be sent to the CRT display. Some boards use VRAM to serialize data.
- The **Attribute Controller** contains one of two color lookup tables (LUTs) that translate color information from the display memory into color information for the CRT display. The first lookup table is controlled via Palette registers of the attribute controller, and the second table is contained in video DACs. Because of the relatively high cost of display memory, a practical display system will typically use a display that supports many more colors than the matching display adapter can simultaneously display. By programming a color lookup table on the display adapter, a programmer can select which subset of the display's colors will be supported for his software.
- The **Video DACs** (Digital to Analog Converters) convert digital color data into an analog signal. They also contain the second color lookup table.
- The **Sequencer** controls the overall timing of all functions on the board. It also contains logic for enabling and disabling color planes.

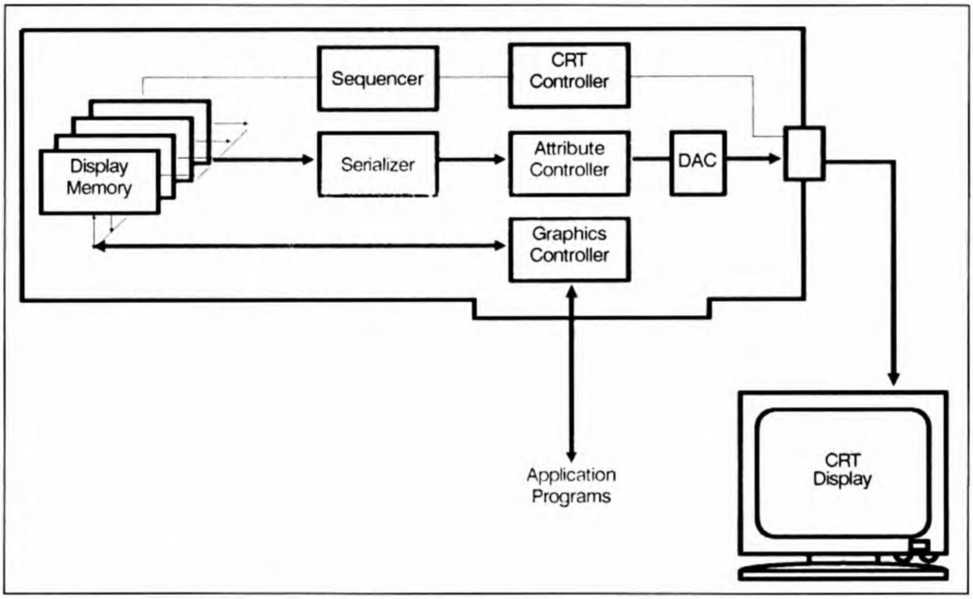


**Figure 2-5. VGA block diagram**

# Display Memory

The VGA contains 256 K (or more) of display memory, divided into four independent 64 K (or 128 K) sections of memory called **color planes**. These memory planes all reside in the same processor memory space. Which color planes are being written to or read from at any time is determined by the settings of several I/O registers.

With all four memory planes residing in the same address space, the processor can write to all four planes (or any combination thereof) with a single memory write cycle. This capability can be very useful for some drawing operations, such as fast screen fills. In other drawing operations, it may be desirable to disable writing to all but a single memory plane. Color planes are enabled and disabled for writing via the Color Plane Write Enable register of the Sequencer.

Since it would not be meaningful for the processor to attempt to read data from more than one source at a time, only one memory plane may be enabled for reading. A color plane is enabled for reading via the Read Plane Read Select register of the Graphics Controller. A special mode is provided, however, to read data from multiple color planes, compare it to some preset reference data, and return status to the processor declaring if the colors matched. The color compare function is useful for finding certain patterns in display memory during operations such as area fills. This mode is controlled by the Color Compare register of the Graphics Controller.

In some operating modes, the organization of display memory will be altered. The best example of this is text mode, where even memory addresses (containing ASCII data) are in memory plane 0, odd memory addresses (containing text attributes) are in memory plane 1, memory plane 2 is reserved for the character generator, and memory plane 3 is unused.

For many operating modes, the 64K address space of the EGA is divided into several display pages. Application software may then control which page is active (being viewed) at any time, and drawing operations can take place in off-screen display memory.

The processor address space used by the EGA and VGA depends on the operating mode. This address space may begin at address A0000, B0000, or B8000, depending on the mode.

## Display Memory in Text Modes

Text mode displays have been in common use much longer than graphics displays, and are still very useful in applications which do not require graphics (or in which simple block graphics will suffice). Text modes place a much lower burden on the system processor, which only has to manipulate ASCII character codes rather than individual pixels.

In standard text modes, the display screen is divided into 25 lines of text, with either 40 or 80 columns of text per line. In 40-column modes, 1000 characters can be displayed on the screen; in 80-column modes, 2000 characters can be displayed (see Figure 2-6). Two bytes of display memory are used to define each character; the first byte, mapped at an even memory address, contains the ASCII character code, and the second byte, mapped at an odd memory address, contains color information called the **Character Attribute**. 2000 bytes of display memory are needed to define one 40 column page, or 4000 bytes to define one 80 column page. A page of display memory is 4096 bytes long, leaving 96 bytes unused at the end of each page.

> Preserving display memory during a Mode Select
>
> BIOS mode select functions will optionally preserve the contents of display memory if the desired mode number is ORed with the value 80h before the BIOS call is made. This capability is limited in text modes, however, since these modes utilize display memory plane 2 for storage of character generators. It is therefore not possible to enter and exit a text mode without corrupting at least part of the display memory.

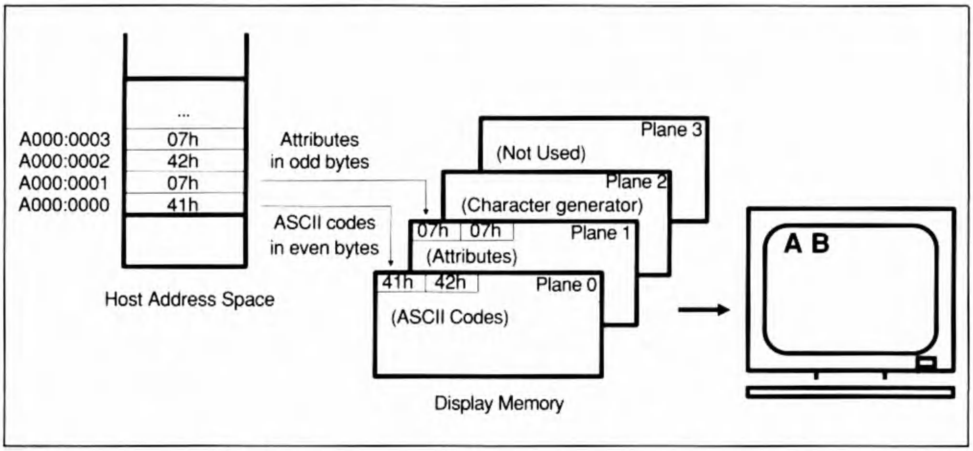


Figure 2-6. Display memory format—text modes

## *Character Generators*

To convert an ASCII character code into an array of pixels on the screen, a translation table or **Character Generator** is used. On older display adapters such as MDA and CGA, the character generator is located in ROM (Read Only Memory.) The VGA does not use a character generator ROM; instead, character generator data is loaded into plane 2 of the display RAM. This feature makes it easy for custom character sets to be loaded. Multiple character sets (up to 8) may reside in RAM simultaneously. A set of BIOS services are available for easy loading of character sets. Figure 2-7 illustrates how character codes are used as an index into a character generator.

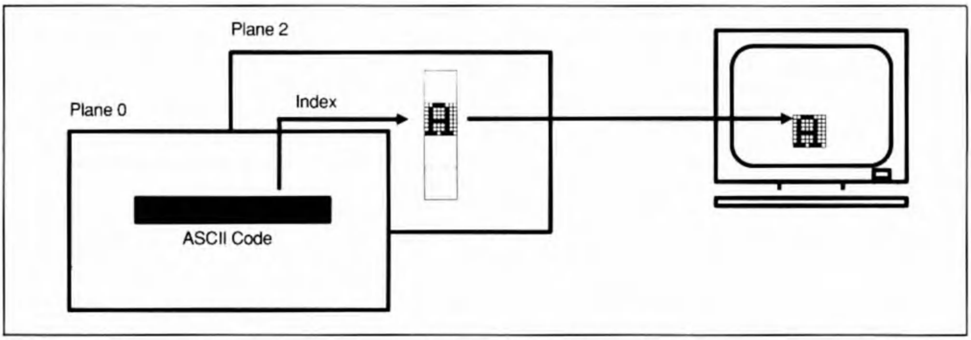


Figure 2-7. Character code as index into character generator

Either one or two character generators may be active, giving the VGA the capability to display up to 512 different characters on the screen simultaneously. When two character generators are active, a bit in each character attribute byte selects which character set will be used for that character. A register in the Sequencer is used to select the two active character generators.

Character width is fixed at eight pixels. Character height is selectable from 1 to 32 pixels through an output register. Figure 2-8 illustrates how a character generator is designed.

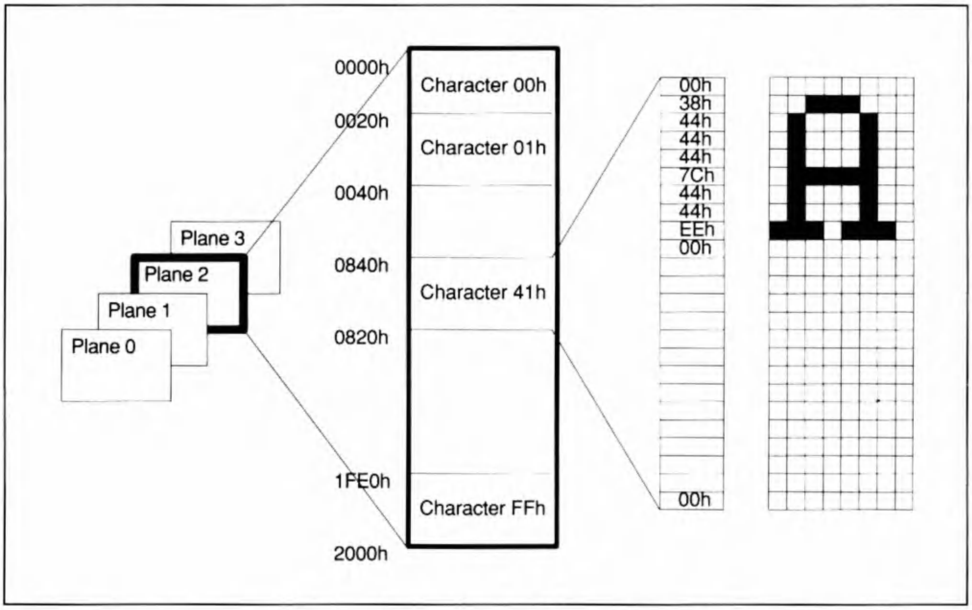


**Figure 2-8. Character generator format**

The location of character generators in memory is shown in Table 2-1. Regardless of the character height which is being used, characters always begin on 32-byte boundaries. For instance, the 8 pixel by 14 pixel character set requires 14 bytes per character, so 18 bytes per character go unused in the character generator.

Table 2-1. Location of RAM-resident character generators

| | |
|---|---|
| **Character Map A** | **Character Map E** |
| 0000h to 001Fh - Char. 0 | 2000h to 201Fh - Char. 0 |
| 0020h to 003Fh - Char. 1 | 2020h to 203Fh - Char. 1 |
| 0040h to 005Fh - Char. 2 | 2040h to 205Fh - Char. 2 |
| 1FE0h to 1FFFh - Char. 255 | 3FE0h to 3FFFh - Char. 255 |
| **Character Map B** | **Character Map F** |
| 4000h to 401Fh - Char. 0 | 6000h to 601Fh - Char. 0 |
| 5FE0h to 5FFFh - Char. 255 | 7FE0h to 7FFFh - Char. 255 |
| **Character Map C** | **Character Map G** |
| 8000h to 801Fh - Char. 0 | A000h to A01Fh - Char. 0 |
| 9FE0h to 9FFFh - Char. 255 | BFE0h to BFFFh - Char. 255 |
| **Character Map D** | **Character Map E** |
| C000h to C01Fh - Char. 0 | E000h to E01Fh - Char. 0 |
| DFE0h to DFFFh - Char. 255 | FFE0h to FFFFh - Char. 255 |

To learn more about character generators, see the following topics in Chapters 3 and 4:

- The ROM BIOS - Function 11h (Load Character Generator)
- VGA Registers - Sequencer Index 3 (Character Generator Select Register)
- VGA Registers - CRT Controller Index 9 (Maximum Scan Line/Character Height)

## *Text Attributes*

Each ASCII character being displayed on the screen has a corresponding attribute byte to define the colors and other attributes that character will have. The interpretation of text attributes depends on operating mode.

**Standard Color Text Attributes** Figure 2-9 shows the bit definitions for text attribute bytes when operating in a standard color text mode. Bits D0-D2, Foreground Color, select the color for the body of the character. Bits D4-D6, Background Color, select the color for the rest of the character cell.

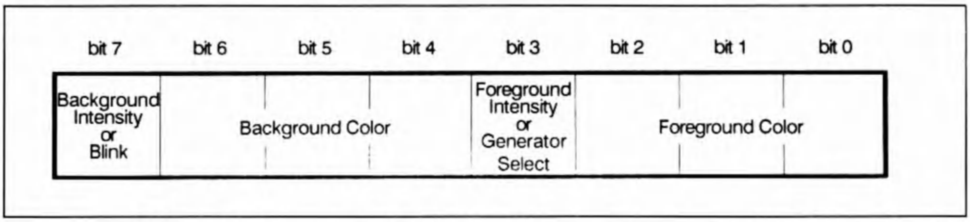


**Figure 2-9.** **Color text attributes**

Attribute bit D3 can be used as a foreground color intensity control, effectively doubling the number of foreground colors from 8 to 16.

If two character sets are being used simultaneously (as defined by the Character Generator Select register of the Sequencer,) bit D3 selects which character set will be used. In this case, the color palette registers of the Attribute Controller should be modified to disable D3 from affecting color.

Attribute bit D7 can be used either as a foreground blink enable, causing the character to blink, or as a background intensity control, doubling the number of background colors from 8 to 16. The function of bit D7 is defined in the mode register of the Attribute Controller. The default setting enables blinking.

Table 2-2 shows the standard colors which are used for both foreground and background.

**Table 2-2.** **Standard color attributes**

| Attribute | Standard Color | Intensified Color |
|---|---|---|
| 000 | Black | Dark Gray |
| 001 | Blue | Light Blue |
| 010 | Green | Light Green |
| 011 | Cyan | Light Cyan |
| 100 | Red | Light Red |
| 101 | Magenta | Light Magenta |
| 110 | Brown | Yellow |
| 111 | Light Gray | White |

**Monochrome Text Attributes** Table 2-3 shows the bit definitions for a monochrome text attribute byte, which is similar in function to a color text attribute byte. Bits D0-D2 control foreground attributes, which can be normal, blanked, or underlined. Bit D3 will intensify the character foreground. Bits D4-D6 can select a reverse video character. Bit D7 can be used as either foreground blink enable or background intensity control; this function is controlled in the Mode Control register of the Attribute Controller. The default setting enables foreground blinking.

As with color attributes, bit D3 can be used to select between two active character sets.

As can be seen from Table 2-3, there are only a small number of valid text attributes in monochrome mode. All attribute values not shown in Table 2-3 should be considered invalid. Use of invalid attributes will create compatibility problems when the software is run on different types of monochrome display adapters (MDA, EGA, VGA, and Hercules.)

**Table 2-3. Monochrome (MDA) text attributes**

| Monochrome Display Attributes | |
|---|---|
| 00000000 | Blank |
| 00000111 | Normal character |
| 10000111 | Blinking character |
| 00001111 | Intensified character |
| 10001111 | Blinking intensified character |
| 00000001 | Underlined character |
| 10000001 | Blink underlined character |
| 00001001 | Intensified, underlined character |
| 10001001 | Blinking, intensified, underlined character |
| 01110000 | Reverse video |
| 11110000 | Blinking reverse video |

It should be noted that if a character is reverse video it cannot be underlined or intensified.

To learn more about text attributes, see the following topics in Chapter 4:

- Function 8 - Read Character and Attribute at Cursor Position
- Function 9 - Write Character and Attribute at Cursor Position
- Function 10h - Set VGA Palette Registers

# Display Memory in Graphics Modes

## *Mode 6 (CGA Two-color Graphics)*

At 640 pixels horizontally and 200 lines vertically, Mode 6 is the highest resolution mode of the CGA adapter. It uses only one bit per pixel (eight pixels per byte). A pixel value of zero displays black, and a pixel value of one displays white. Pixel data is stored in color plane 0. Display data is serialized most significant bit first, so the first bit position in the upper left corner of the screen displays the data in bit D7 of byte 0 of display memory.

Limitations of the 6845 CRT Controller which was used on the CGA resulted in a nonlinearly mapped display memory address space for all graphics modes. This com-

plicates drawing algorithms, since a computation is required to translate between a pixel position on the display screen and a bit position in display memory.

Figure 2-10 illustrates the translation that occurs between the display memory and the display screen. The first half of display memory contains the data for all even numbered CRT scan lines. The second half of display memory contains the data for all odd numbered scan lines. To translate from a pixel position (x,y) on the display screen where x is the horizontal coordinate in the range 0-639 and y is the vertical coordinate in the range 0-199, to a bit position in display memory, use the following formula:

Byte address = 80*(y/2) + (x/8) if y is even
Byte address = 8192 + 80*((y-1)/2) + (x/8) if y is odd
bit position (0-7) = 7 - (x modulo 8)

(The modulo operator is equivalent to taking the remainder of x/8).

## Modes 4 and 5 (CGA Four-color Graphics)

These are the most colorful, as well as most popular, graphics modes of the CGA adapter. The resolution is low; only 320 pixels horizontally by 200 lines vertically. The display memory map uses packed pixels, two bits per pixel, packed four pixels per byte. Pixel data is stored as one plane (planes 0 and 1 chained to form one plane). Display data is serialized most significant bit first, so the first bit position in the upper left of the screen displays the data in bits D7 and D6 of byte 0 of display memory.

As with all CGA graphics modes, the display memory is nonlinearly mapped. A computation is required to translate from a pixel location on the display screen to a bit

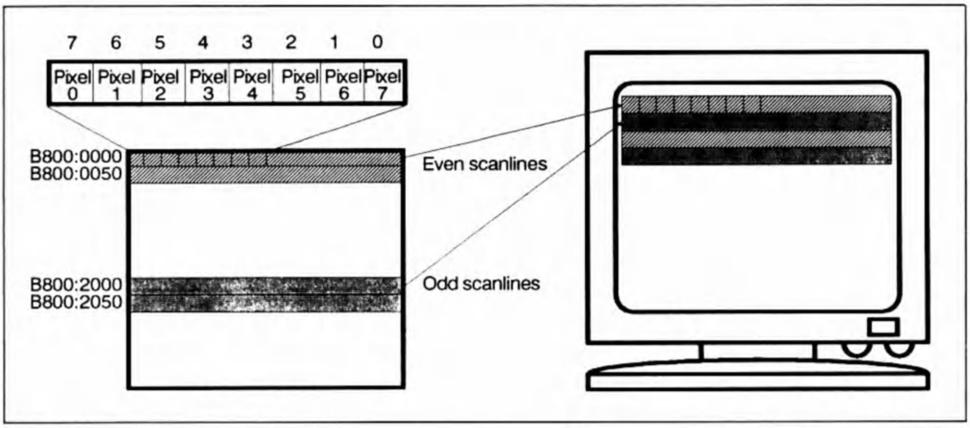


Figure 2-10. Memory map—CGA graphics mode 6

location in display memory. Figure 2-11 illustrates the memory map for modes 4 and 5. The first half of display memory contains the data for all even numbered scan lines. The second half of display memory contains the data for all odd numbered scan lines.

To translate from a pixel location (x,y) on the screen to a bit location in display memory, where x is a horizontal coordinate in the range 0–319 and y is a vertical coordinate in the range 0–199, use the following formula:

Byte address = 80*(y/2) + (x/4)    if y is even
Byte address = 8192 + 80*((y-1)/2) + (x/4)    if y is odd
Bit position (0,2,4,6) = (x modulo 4) * 2

Two standard color sets are supported in modes 4 and 5. A BIOS call (BIOS function 0Bh) is used to select colors. The standard colors for modes 4 and 5 are shown in Table 2-4.

**Table 2-4. Standard colors—modes 4 and 5**

| Pixel Value | Standard Color | Alternate Color |
|---|---|---|
| 00 | Black | Black |
| 01 | Light Cyan | Green |
| 10 | Light Magenta | Red |
| 11 | Intensified White | Brown |

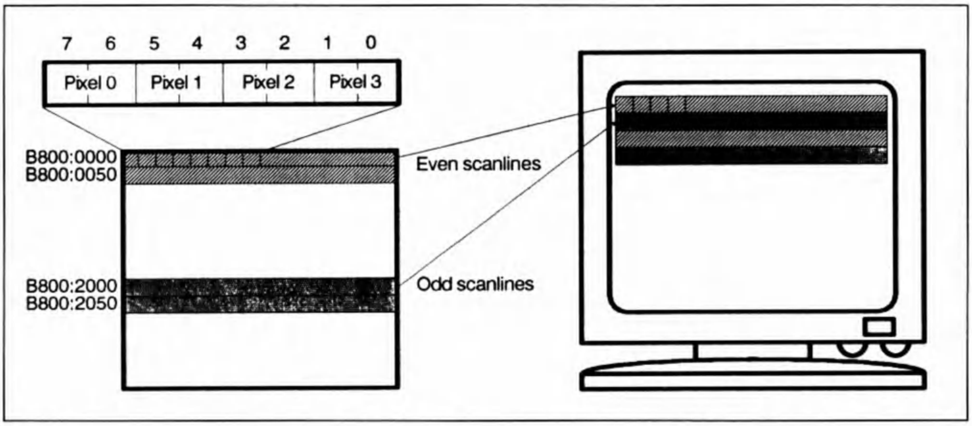


**Figure 2-11. Memory map—CGA graphics modes 4 and 5**

## Mode F - Monochrome Graphics

Mode F, which is unique to EGA and VGA, does not suffer from the nonlinear addressing problems of CGA graphics modes. Resolution is 640 pixels horizontally by 350 pixels vertically. Two color planes are used (planes 0 and 1). Each pixel occupies one bit in each color plane. The four "colors" supported by these two-bit pixels are black, white, intensified white, and blinking. The two color planes are independently enabled and disabled for writing through the Color Plane Write Enable register of the Sequencer.

Organization of the memory is similar to that in Figure 2-12, except that only planes 0 and 1 are used. To translate from a pixel (x,y) on the screen to a bit location in display memory, where x is a horizontal coordinate in the range 0-639 and y is a vertical coordinate in the range 0-349, use the following formula:

Byte address = y*80 + x/8
Bit position (0-7) = 7 - (x modulo 8)

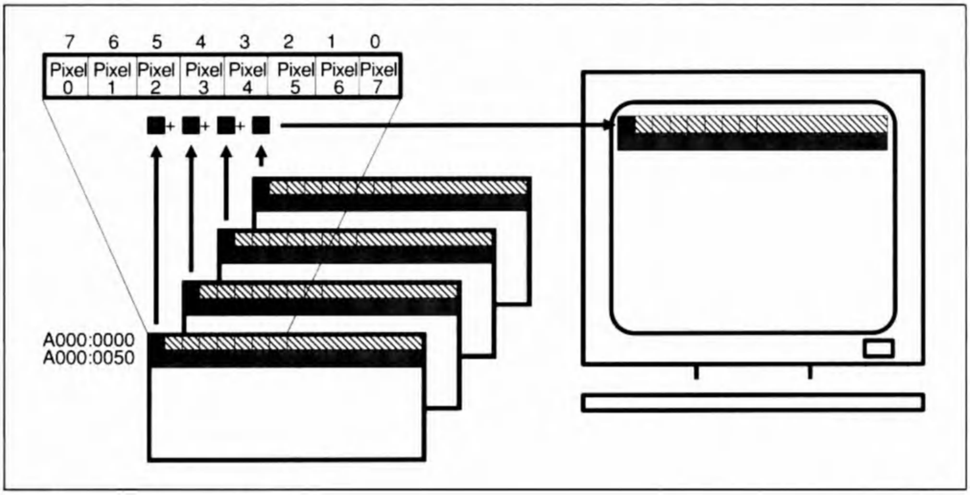


**Figure 2-12. Memory map—Planar modes (Dh, Eh, Fh, 10h, 11h, 122h)**

## Mode 10 - Enhanced Color Graphics

Mode 10, which is unique to EGA and VGA, is the most popular mode for new color graphics applications. Resolution is 640 pixels horizontally by 350 lines vertically. All four color planes are used. Color planes are independently enabled and disabled for writing through the Color Plane Write Enable register of the Sequencer. Each pixel

occupies one bit in each color plane. These four-bit pixels permit 16 simultaneous colors to be displayed.

Figure 2-12 illustrates the memory map planar modes such as mode 10h. To translate from a pixel (x,y) on the screen to a bit location in display memory, where x is a horizontal coordinate in the range 0–639 and y is a vertical coordinate in the range 0–349, use the following formula:

Byte address = y*80 + x/8
Bit position (0-7) = 7 - (x modulo 8)

## Modes D and E (Sixteen-color Graphics)

Modes D and E are very similar to mode 10 in operation, differing only in screen resolution. Mode D operates at a resolution of 320 pixels horizontally by 200 pixels vertically. Mode E operates at a resolution of 640 pixels horizontally by 200 pixels vertically. These modes have not become popular because of the limited resolution they offer.

## Mode 11 (Two-color Graphics)

Mode 11 is unique to the VGA adapter. Resolution is 640 pixels horizontally by 480 pixels vertically, but only two colors are supported. Display data is stored in plane 0, and the other planes are unused. Each pixel occupies one bit in display memory.

Display memory is similar to that shown in Figure 2-12, except that only one plane is used. To translate from a pixel (x,y) on the screen to a bit location in display memory, where x is a horizontal coordinate in the range 0–639 and y is a vertical coordinate in the range 0–479, use the following formula:

Byte address = (y*80) + (x/8)
Bit position (0-7) = 7 - (x modulo 8)

## Mode 12 (Sixteen-color Graphics)

Mode 12, which is unique to VGA, is similar to mode 10 hex except that the vertical resolution is expanded from 350 lines to 480 lines. All four color planes are used, and 16 simultaneous colors are supported. The organization of memory is the same as in Figure 2-12.

## Mode 13 (256-color Graphics)

Mode 13, which is also unique to VGA, allows 256 simultaneous colors to be used at a low resolution (320 pixels horizontally by 200 lines vertically.) Memory is linearly mapped as shown in Figure 2-13. To translate from a pixel (x,y) on the screen to a bit

location in display memory, where x is a horizontal coordinate in the range 0–319 and y is a vertical coordinate in the range 0–199, use the following formula:

Byte address = (y*320) + x

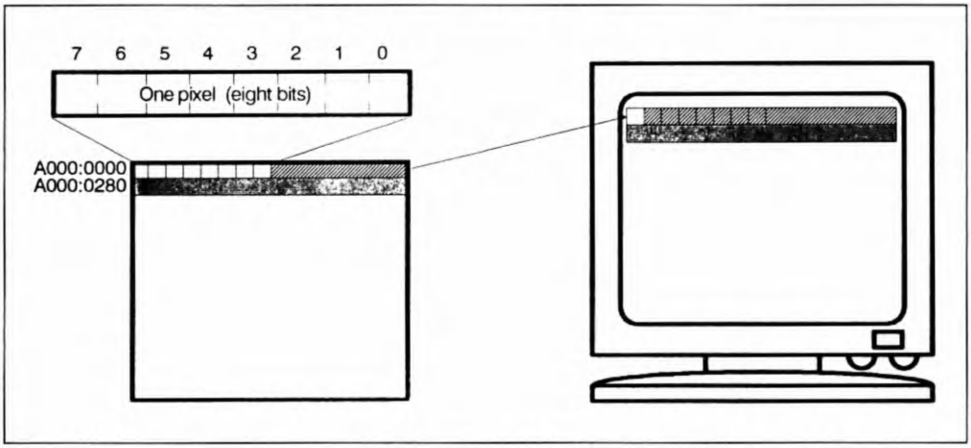


**Figure 2-13. Memory map—VGA graphics mode 13**

# The Graphics Controller

The Graphics Controller resides in the data path between the processor and display memory. In its default state, the Graphics Controller is transparent. Data can be written to and read from display memory with no alterations. The Graphics Controller can, however, be programmed to assist in drawing operations by performing tasks that would otherwise have to be performed by the main processor.

## Processor Read Latches

Each time the system processor reads data from display memory, the data is also latched into the VGA's on-board read latches. During write cycles, the data in these read latches can be logically combined with write data from the processor. If properly used, this function can assist the processor in performing drawing operations. While the processor can only read data from one plane at a time, the read latches latch data from all four planes simultaneously. This can be used to quickly copy data from one region of display memory to another.

## Logical Unit

During display memory write cycles, the Graphics Controller can perform any of the following functions on the write data:

- Write data unmodified
- Logical OR write data with data in read latches
- Logical AND write data with data in read latches
- Logical XOR write data with data in read latches
- ROTATE write data

Logical AND/OR/XOR functions are useful for adding and removing foreground display elements over the background (such as graphics cursors and sprites). Data rotation is useful when performing block transfers of non-byte-aligned data.

The function of the Graphics Controller during write operation is illustrated in Figure 2-14. To learn more about the proper use of the read latches and logical unit, see:

- Data Rotate and Function Select register, index 3, of the Graphics Controller - Chapter 3
- Mode register, index 5 of the Graphics Controller - Chapter 3

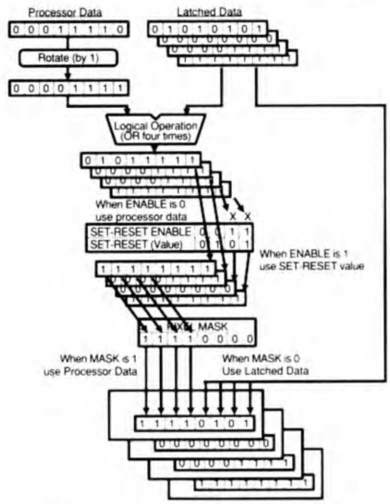


**Figure 2-14. Graphics controller write operation**

## Color Compare

During processor read cycles, the Graphics Controller can perform a function called Color Compare, which is useful for drawing algorithms such as *Flood Fill* where a specific screen color or change in color must be detected. Using normal display memory read cycles, the processor may only interrogate one color plane at a time. With Color Compare, however, the processor enters a reference color into a register in the Graphics Controller. During a read cycle, the Graphics Controller compares the data in all four planes (or any selected subset of the four planes) against the reference color and indicates whether a color match was found.

Color Compare provides the ability to search display memory for an object of a specific color, especially when used with the 8086 REP SCASB instruction.

To learn more about the Color Compare function, see:

- The Color Compare Register, index 2 of the Graphics Controller - Chapter 3
- The Color Don't Care Register, index 7 of the Graphics Controller - Chapter 3
- The Mode Register, index 5 of the Graphics Controller - Chapter 3

## Data Serializer

The Data Serializer captures the data read from display memory during display refresh cycles and converts it into a serial bit stream to drive the CRT display. Display data is serialized most significant bit first. Some boards use VRAM for their display memory, and in such cases VRAM is used to serialize data.

# The Attribute Controller and DACs

The Attribute Controller and DAC registers determine which colors will be displayed for both text and graphics. The heart of the Attribute Controller is a **color lookup table (LUT)** that in planar modes translates four-bit color codes from display memory into six-bit color codes. These color codes are combined with a **Color Select register** value to form 8-bit codes that are fed to the video DAC. A second color lookup table, internal to the DACs, converts this eight-bit code into an eighteen-bit code (six bits each for the red, green and blue guns). In 256-color modes, attribute controller registers are setup to pass the eight-bit codes from display memory directly to video DAC, without translation.

As part of a BIOS mode select operation, the color lookup tables are initialized with data appropriate for that mode. For monochrome modes, the tables are initialized to display only two colors. For CGA modes, the tables are initialized to support the limited colors available with that adapter. For EGA and VGA modes, the tables are initialized to support the richer colors of those adapters. Application software may at any

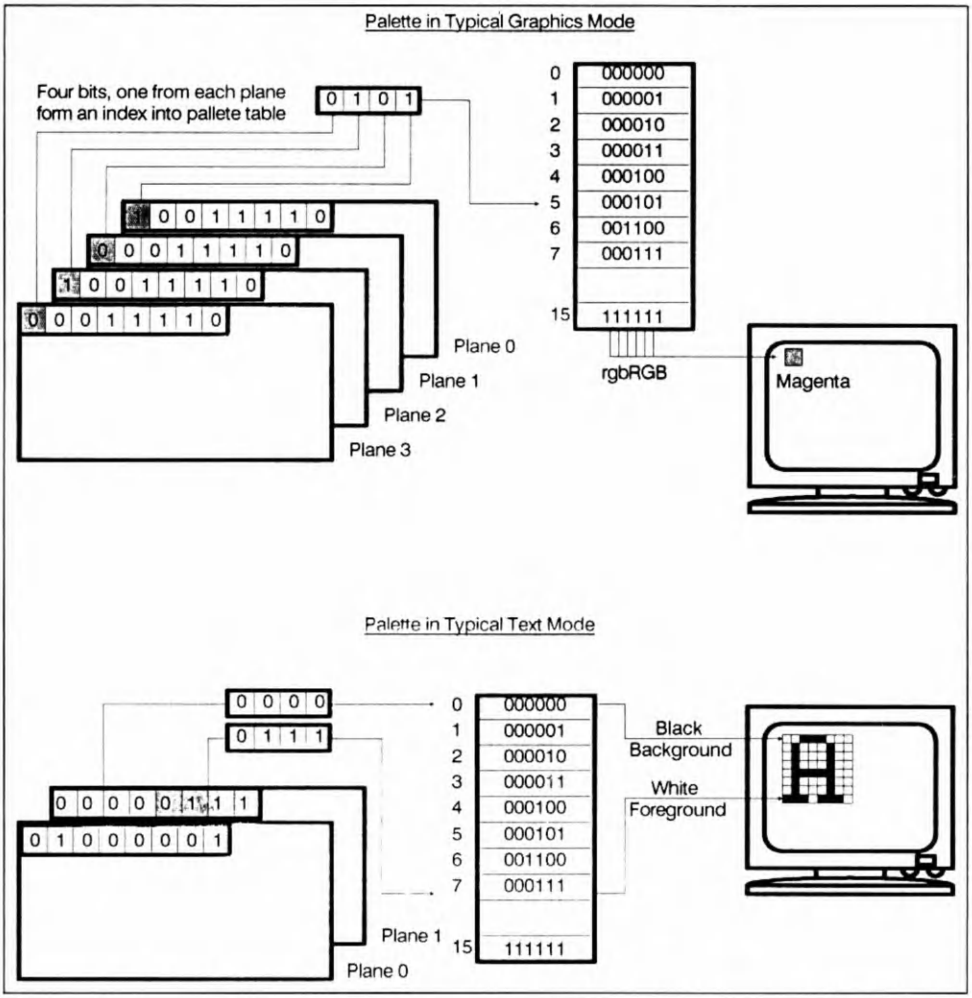


**Figure 2-15. Color lookup tables**

time redefine the color palette by reprogramming the Attribute Controller and/or the DAC registers.

Figure 2-15 illustrates the function of the color lookup tables during a screen refresh cycle for a typical planar mode. In the diagram a pixel color value of 0101 (binary 5) has been read from the color planes. This color value is used as an address to select a register in the color lookup table. Register 5 in the lookup table contains the binary data value 000101, which results in a magenta pixel on the screen (assuming default values in DAC registers).

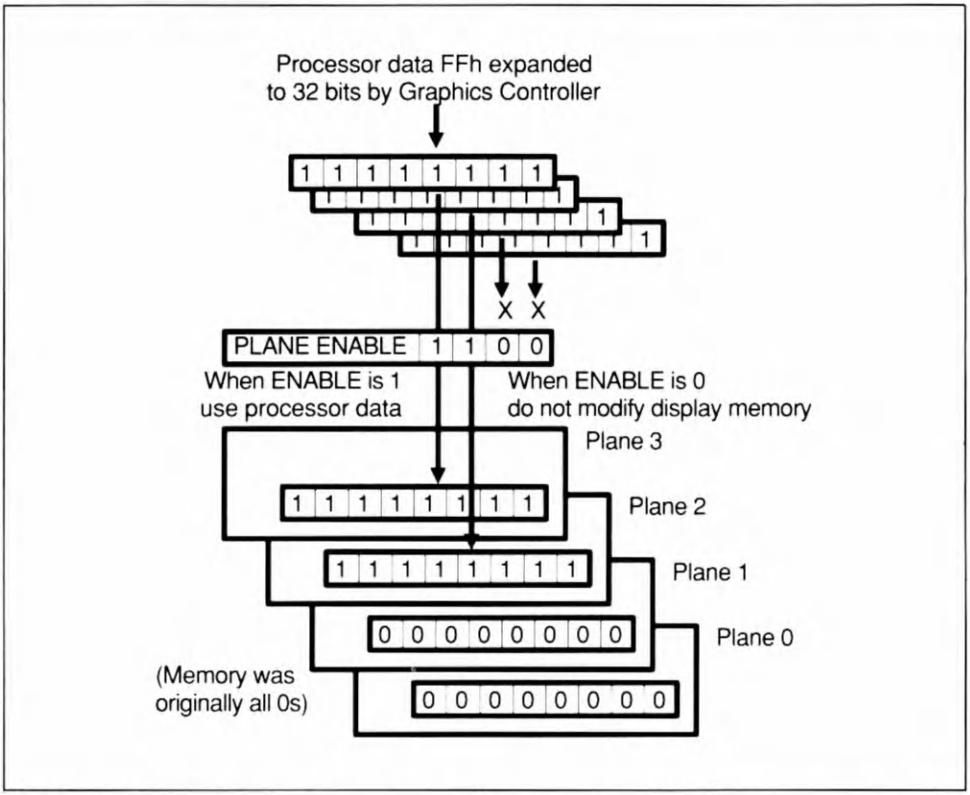


**Figure 2-16. Plane write enable function of the sequencer**

Note that the color (attribute) is represented differently in text modes than in graphics modes.

To learn more about the color lookup tables, see:

- Function 0Bh - Set CGA Color Palette - Chapter 4
- Function 10h - Set EGA Palette Registers - Chapter 4
- The Attribute Controller registers - Chapter 3
- The Video DAC registers - Chapter 3

# The CRT Controller

Most registers of the CRT Controller are set by the BIOS to define CRT timing and should not be modified. Other CRT Controller registers define cursor shape and position and perform vertical scrolling.

The VGA CRT Controller is functionally very similar to the Motorola 6845 CRT Controller used on the MDA, CGA and Hercules display adapters, but it is not register compatible. Software which directly addresses 6845 registers will not, in general, run properly on the EGA or VGA (or vice versa.) To make matters even more complicated, the CRT Controller of the VGA is not identical to that of the EGA. For the sake of compatibility, it is a good idea to use the BIOS functions when possible in order to avoid making direct accesses to registers in the CRT Controller.

## The Sequencer

The Sequencer generates the dot and character clocks that control display refresh timing. It controls the timing of display memory read and write cycles, and generates wait states to the processor when necessary.

The Sequencer also contains logic for enabling and disabling processor access to specific color planes. It is this function that makes the Sequencer interesting to programmers. Its action is illustrated in Figure 2-16.

# 3

# *VGA Registers*

# Introduction

The VGA contains more than 60 registers. To avoid monopolizing a large piece of the processor I/O space, the registers of the VGA are multiplexed into a small number of I/O addresses. In most cases, register access is a two-step procedure of selecting a register through one I/O port, then reading or writing data through a second I/O port.

The I/O addresses used depend on the operating mode. To achieve compatibility with both the MDA and CGA display adapters, some I/O addresses must be mapped differently for color modes than for monochrome modes. Tables 3-1 and 3-2 list the I/O addresses used in color and monochrome modes.

**Table 3-1. VGA monochrome I/O map**

| I/O Address | Registers |
|---|---|
| 3CCh | Miscellaneous Output register (read-only) |
| 3C2h | Miscellaneous Output register (write-only) |
| | Input Status register 0 (read-only) |
| 3BAh | Feature Control register (write-only) |
| | Input Status register 1 (read-only) |
| 3C4h,3C5h | Sequencer |
| 3B4h,3B5h | CRT Controller |
| 3CEh,3CFh | Graphics Controller |
| 3C0h,3C1h | Attribute Controller |
| 3C3h/46E8 | VGA Enable |
| 3C6h,3C7h,3C8h,3C9h | VGA Video DAC |

**Table 3-2. VGA color I/O map**

| I/O Address | Register |
|---|---|
| 3CCh | Miscellaneous Output register (read-only) |
| 3C2h | Miscellaneous Output register (write-only) |
| | Input Status register 0 (read-only) |
| 3DAh | Feature Control register (write-only) |
| | Input Status register 1 (read-only) |
| 3C4h,3C5h | Sequencer |
| 3D4h,3D5h | CRT Controller |
| 3CEh,3CFh | Graphics Controller |
| 3C0h,3C1h | Attribute Controller |
| 3C3h/46E8h | VGA Enable |
| 3C6h,3C7h,3C8h,3C9h | VGA Video DAC |

VGA I/O register addresses can be grouped logically according to function. The CRT Controller, Graphics Controller, Attribute Controller and Sequencer each have their own set of addresses. Later sections will describe each of these in detail. The remaining registers, which do not belong to any of the major functional blocks, are described in the next section "Control Registers"

Most of the registers of the VGA are not of practical interest to programmers. Once they are properly initialized by the BIOS for the display mode being used, most registers require no further servicing and can effectively be ignored.

With older adapters, including EGA, there can be danger involved in modifying timing registers. Some inexpensive displays will literally burn up if driven with improper timing as a result of improper register settings. This is less of a problem with VGA only because today's newer displays tend to be sturdier in this regard.

A small number of VGA registers can be used by the programmer to perform such functions as cursor control, panning and scrolling, split screen displays, and others.

When referencing the bits of a register or memory byte, conventions shown in Figure 3-1 will be used.

| MSB | | | | | | | LSB | |
|---|---|---|---|---|---|---|---|---|
| D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | |
| 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 78h |

**Figure 3-1. Bit annotation convention**

Index registers retain their current value until modified. If a particular register is going to be written to or read from repetitively, the index only needs to be set once; the data register can then be written or read as many times as needed.

Before modifying any VGA registers, read the register description provided in this book carefully. In general, the following rules must be adhered to when modifying registers:

Before modifying some registers of the Sequencer, the Sequencer must be placed into a reset state using its Reset register. While the sequencer is in a reset state, all functions of the VGA, including memory refresh, are halted.

Before modifying the palette registers of the Attribute Controller, the PALETTE ADDRESS SOURCE bit must be reset in the Attribute Controller index register. While this bit is reset, the display will be blanked.

The I/O addresses of several VGA registers depend on the type of display being used (monochrome or color.)

CRT Controller registers that are involved in video timing require special care since incorrect register values may cause damage to the display. Any routine that loads video timing registers must be written in assembly language. A high level language (such as C or Pascal) will execute too slowly, possibly resulting in invalid display timing for an unacceptable length of time. It is also recommended that the display be blanked while timing registers are modified.

For the majority of applications which use the VGA in one of its standard operating modes, initialization of video timing registers should be left to the BIOS. Access to these registers can then be avoided entirely.

The important criteria to be considered when modifying VGA registers are summarized by the following pseudocode:

```
if (Miscellaneous Register is being modified)
                No special care needed, write directly to register
                (if clock changes then do synchronous reset on Sequencer)

else if (CRT Controller register is being modified)
                {
                is CRTC in monochrome mode (address 3B4) or color mode (address 3D4)?
                is it locked register? (unlock if so)
                is it dangerous register? (may want to turn video off)
                output register index to address 3B4 or 3D4
                output register data to address 3B5 or 3D5
                }

else if (Attribute Controller register is being modified)
                {
                Is the current mode color or monochrome?
                Reset attribute controller Index/Data flip-flop by reading address 3BA or 3DA.
                If (this is a color palette register)
                                {
                                output register index, with PALETTE ADDRESS SOURCE bit clear, to
                                    address 3C0
                                output register data to address 3C0
                                output 20h to address 3C0 to re-enable video
                                }
                else
                                {
                                output register index, with PALETTE ADDRESS SOURCE bit set, to
                                    address 3C0
                                output register data to address 3C0
                                }
                }

else if (Graphics Controller register is being modified)
                {
                output register index to address 3CE
                output register data to address 3CF
                }

else if (Sequencer register is being modified)
                {
                if (Clock Mode Register)
                                {
```

```
                    output 0 to index register (3C4)
                    output 01 to data register (3C5) to synchronously halt the
                        sequencer
                    output register index to index register (3C4)
                    output register data to data register (3C5)
                    output 0 to index register (3C4)
                    output 3 to data register (3C5) to re-enable sequencer
                    }
    else
                    {
                    output register index to index register (3C4)
                    output register data to data register (3C5)
                    }
    }
```

# Control Registers

## Miscellaneous Output Register (I/O Address Write 3C2h, Read 3CCh)

Care must be taken when modifying this register because it controls, among other things, the polarity of the sync outputs to the display and the video clock rate. There are two register bits here, however, that may be of interest to some programmers.

D7 - Vertical Sync Polarity
D6 - Horizontal Sync Polarity
D5 - Odd/Even Page
D4 - Disable Video
D3 - Clock Select 1
D2 - Clock Select 0
D1 - Enable/Disable Display RAM
D0 - I/O Address Select

**Sync Polarity** bits are set as shown in Table 3-3 for VGA displays that use sync polarity to determine screen resolution. Many newer multiple frequency displays are insensitive to sync polarity.

The **Disable Video** bit should not be used as a general purpose display on/off control, since it disables CRT sync signals as well as video output. It can be used to permit another device access to the display through the feature connector.

**Table 3-3. Sync polarity vs. vertical screen resolution**

| D7 D6 | Resolution |
|---|---|
| 0 0 | Invalid |
| 0 1 | 400 lines |
| 1 0 | 350 lines |
| 1 1 | 480 lines |

**Enable/Disable Display RAM** can be used to disable the display memory from being written or read by the host.

**I/O Address Select**, when set to zero, selects the monochrome I/O address space (3Bx). When set to one, it selects the color I/O address space (3Dx).

## Input Status Register 0 (I/O Address 3C2, Read only)

D7 - Vertical Retrace Interrupt Pending
D6 - Feature Connector Bit 1
D5 - Feature Connector Bit 0
D4 - Switch Sense
D0 to D3 - Unused

**Vertical Retrace Interrupt Pending** can be polled by an interrupt handler to determine if vertical retrace was the cause of an interrupt. It is cleared through the Vertical Retrace End register in the CRT Controller. Vertical Retrace is available as an interrupt source on IRQ2 on most VGA implementations.

## Input Status Register 1 (I/O Address 3BAh/3DAh, Read only)

D7 - Unused
D6 - Unused
D5 - Diagnostic
D4 - Diagnostic
D3 - Vertical Retrace
D2 - Unused
D1 - Unused
D0 - Display Enable

**Vertical Retrace** gives the real-time status of the vertical sync signal (1 = sync pulse active).

**Display Enable** gives the real-time status of the display blanking signal.

For EGA, the **Diagnostic** bits are the only means of reading back the contents of the color lookup table in the Attribute Controller. For VGA, these registers can be read directly.

## VGA Enable Register (I/O Address 3C3h/46E8h)

D7-D1 - Reserved
D0 - VGA Enable/Disable (3C3h only)

Register 3C3h enables and disables reads and writes to VGA memory and I/O (except this register). On the IBM add-in VGA (not the motherboard resident VGA),

and on some SuperVGAs, an alternate I/O address 46E8h is used instead of or in addition to 3C3h. Since support for register 46E8 is not well standardized, it is best to use BIOS function 12h, subfunction 32h, (Enable/Disable VGA Access).

# The CRT Controller Registers

Two I/O addresses are used by the CRT Controller. The first address is an index register which is used to select one of the 25 internal registers of the CRT Controller (see Table 3-4). The second address is used to read data from or write data to the selected register.

The I/O addresses of the CRT Controller depend on the operating mode. In monochrome modes, the index register is mapped at I/O address 3B4 and the data register is at address 3B5. In color modes, the index register is at address 3D4 and the data register is at address 3D5.

**Table 3-4. CRT Controller registers**

| Index | Register |
|---|---|
| 0 | Horizontal Total |
| 1 | Horizontal Display Enable |
| 2 | Start Horizontal Blanking |
| 3 | End Horizontal Blanking |
| 4 | Start Horizontal Retrace |
| 5 | End Horizontal Retrace |
| 6 | Vertical Total |
| 7 | Overflow |
| 8* | Preset Row Scan |
| 9* | Maximum Scan Line / Text Character Height |
| 0Ah* | Cursor Start |
| 0Bh* | Cursor End |
| 0Ch* | Start Address (High Byte) |
| 0Dh* | Start Address (Low Byte) |
| 0Eh* | Cursor Location (High Byte) |
| 0Fh* | Cursor Location (Low Byte) |
| 10h | Vertical Retrace Start |
| 11h | Vertical Retrace End |
| 12h | Vertical Display Enable End |
| 13h* | Offset Register/Logical Screen Width |
| 14h* | Underline Location |
| 15h | Start Vertical Blanking |
| 16h | End Vertical Blanking |
| 17h | Mode Control |
| 18h* | Line Compare |

Many of the compatibility problems that arise between VGA and CGA or MDA are due to the register differences between the VGA and the 6845 CRT Controller that is used on the other adapters. The differences are summarized in Table 3-5.

**Table 3-5. VGA CRT Controller vs. 6845 CRT Controller**

| Index | 6845 | VGA |
|---|---|---|
| 2 | Horizontal Sync Position | Start Horizontal Blanking |
| 3 | Sync Width | End Horizontal Blanking |
| 4 | Vertical Total | Start Horizontal Retrace |
| 5 | Vertical Total Adjust | End Horizontal Retrace |
| 6 | Vertical Displayed | Vertical Total |
| 7 | Vertical Sync Position | Overflow |
| 8 | Interface Mode/Skew | Preset Row Scan |

Most of the registers of the CRT Controller are used to define CRT timing parameters, and should not be modified. Relation of the timing registers is summarized in Figure 3-2. Registers that may be of interest to the programmer are marked (*).

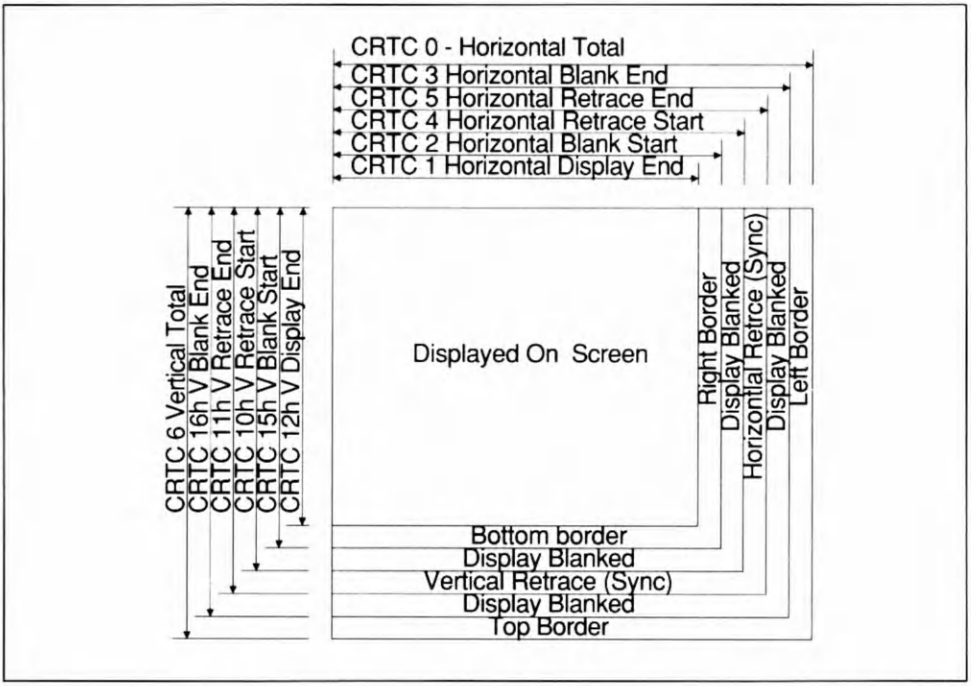


**Figure 3-2. CRTC Timing registers**

## Index 0 - Horizontal Total

Total number of characters in horizontal scan minus five (including blanked and border characters).

## Index 1 - Horizontal Display Enable

Total number of characters displayed in horizontal scan minus one.

## Index 2 - Start Horizontal Blanking

Character at which blanking starts.

## Index 3 - End Horizontal Blanking

D7 - Test
D6 - Skew Control
D5 - Skew Control
D0 to D4 - End Blanking

**End Blanking** is five LSB bits of six-bit value, which define the character at which blanking stops. The MSB bit of this value is in register index 5.

## Index 4 - Start Horizontal Retrace

Character at which horizontal retrace starts.

## Index 5 - End Horizontal Retrace

D7 - End horizontal Blanking Bit 5
D6 - Horizontal Retrace Delay
D5 - Horizontal Retrace Delay
D0 to D4 - End Horizontal Retrace

**End Horizontal Retrace** defines the character at which horizontal retrace ends.

## Index 6 - Vertical Total

Total number of horizontal scan lines minus two (including blanked and border characters). MSB bits of this value are in register index 7.

## Index 7 - Overflow Register

Several of the CRT timing registers of the EGA and VGA are nine-bit registers. The Overflow register is a collection of the ninth bits (most significant bit, or D8) from these registers. The VGA has an added tenth (D9) bit to some registers.

In most cases, the Overflow register can be ignored after the mode select. Unfortunately, there is one bit in the Overflow register (the Line Compare bit) that may be useful for some applications. Great care must be taken to preserve the contents of other bits if this bit is used.

D7 - Vertical Retrace Start (Bit 9)
D6 - Vertical Display Enable End (Bit 9)
D5 - Vertical Total (Bit 9)
D4 - Line Compare (Bit 8)
D3 - Start Vertical Blank (Bit 8)
D2 - Vertical Retrace Start (Bit 8)
D1 - Vertical Display Enable End (Bit 8)
D0 - Vertical Total (Bit 8)

## Index 8 - Preset Row Scan

D7 - Unused
D6 - Byte Panning Control
D5 - Byte Panning Control
D0 to D4 - Preset Row Scan

**Byte Panning Control** is used to control byte panning. This register together with Attribute Controller register 13 (Horizontal Pixel Panning) allows for up to 31 pixels of panning in double word modes (no such modes were defined as of this writing).

**Preset Row Scan** is used for smooth scrolling in text mode, so that character rows can be scrolled up or down one pixel at a time. For the topmost row of text on the screen, this register defines which character scan line will be the first line displayed (so that the top character row shows only partial characters). Smooth scrolling is achieved by slowly incrementing or decrementing the value of this register.

## Index 9 - Maximum Scan Line/Character Height

IBM named this register Maximum Scan Line, but Text Character Height is more appropriate. Maximum Scan Line means the number of scan lines per text character, which is also equal to the pixel height of a character. It is used for text modes only. Character height is one greater than the value of this register.

D7 - Double Scan
D6 - Bit D9 of Line Compare Register
D5 - Bit D9 of Start Vertical Blank Register
D4-D0 - Maximum Scan Line

**Double Scan** enables the double scan mode of the VGA. This mode is used in CGA-compatible graphics modes that have a vertical resolution of 200 scan lines vertically. When double scanning is enabled, each scan line will be displayed twice, increasing the number of scan lines on the display from 200 lines to 400 lines.

Bits D6 and D5 are overflow bits from other registers, located here due to a lack of space in the Overflow register.

## Index 0Ah - Cursor Start

D7,D6 - Reserved (0)
D5 - Cursor Off
D4-D0 - Cursor Start

**Cursor Start** determines at which character row scan the cursor will begin. Together with the Cursor End register, this register defines the size of the cursor with respect to a character cell.

**Cursor Off** disables the cursor display.

Support for Vertical Interrupt varies for different manufacturers. The IBM VGA on system boards provides full support for Vertical Interrupt. CRTC register 11h, bit 5, is used to enable and disable IRQ2. On IBM add-on VGA boards, however, is not supported (IRQ2 trace is not connected to the bus). Many SuperVGA adapters have elected not to follow IBM, and provide full support for Vertical Retrace Interrupt. Care must be taken to keep this interrupt disabled on such VGAs, since it may interfere with some Network Controllers on AT systems (Network Controllers on PS/2 systems do not have this problem).

## Index 0Bh - Cursor End

D7 - Reserved
D6,D5 - Cursor Skew
D4-D0 - Cursor End

This register is the companion to index 0Ah (Cursor Start). It determines the character scan line at which the cursor display will stop.

**Cursor Skew** places a skew on the cursor relative to the character clock. This allows the cursor to be displayed one or two characters to the right of the character specified in CRTC registers 0Eh and 0Fh.

## Index 0Ch - Start Address (High Byte)

## Index 0Dh - Start Address (Low Byte)

This 16-bit register defines the address in display memory of the data that will be displayed in the upper left corner of the screen (starting position). This register can be used to pan an image on the screen, or move between display pages in memory. It also plays a key role in establishing a split screen (see the Line Compare register for details).

## Index 0Eh - Cursor Location (High Byte)

## Index 0F - Cursor Location (Low Byte)

This 16-bit register defines the position of the cursor on the screen. When the screen refresh memory address equals the Cursor Location register, the cursor will be displayed on the screen.

Support for Vertical Interrupt varies for different manufacturers. The IBM VGA on system boards provide full support for Vertical Interrupt. CRTC register 11h, bit 5, is used to enable and disable IRQ2. On IBM add-on VGA boards, however, is not supported (IRQ2 trace is not connected to the bus). Many SuperVGA adapters have elected not to follow IBM, and provide full support for Vertical Retrace Interrupt. Care must be taken to keep this interrupt disabled on such VGAs, since it may interfere with some Network Controllers on AT systems (Network Controllers on PS/2 systems do not have this problem).

## Index 10h - Vertical Retrace Start

Eight LSB bits of ten-bit value, which determine scan line at which vertical retrace starts. The other two bits are in CRTC register index 7.

## Index 11h - Vertical Retrace End

D7 - Write Protect CRTC Registers 0 to 7
D6 - Refresh Cycle Select
D5 - Enable Vertical Interrupt (when 0)
D4 - Clear Vertical Interrupt (when 0)
D0 to D3 - Vertical Retrace End

**Vertical Retrace End** defines four LSB bits of the scan line at which vertical retrace ends. Note that this limits retrace to a maximum of 15 scan lines.

## Index 12h - Vertical Display Enable End

Eight LSB bits of ten-bit value which define scan line minus one at which the display ends. The other two bits are in CRTC register index 7.

## Index 13h - Offset/Logical Screen Width

IBM named this the Offset register, but a better name for this register would be Logical Screen Width. In graphics modes, it defines the logical distance, in either 16-bit words or 32-bit double words, between successive scan lines. In other words, if the screen refresh data for scan line n begins at memory address m, refresh data for scan line n + 1 will begin at address m + offset. In text modes, the offset is the logical increment between successive character rows.

## Index 14h - Underline Location Register

In monochrome text mode only, Underline Location defines which line of a character cell will be illuminated when the underline attribute is set. This register is set during a BIOS mode select operation according to the font size being used.

D7 - Reserved
D6 - Double Word Mode
D5 - Count by 4
D0 to D4 - Underline Location

## Index 15h - Start Vertical Blanking

Eight LSB bits of ten-bit value minus one which define at which scan line the vertical blanking starts. The other two bits are in CRTC registers index 7 and 9.

## Index 16h - End Vertical Blanking

Eight LSB bits of a value which determine the scan line after which vertical blanking ends.

## Index 17h - Mode Control Register

D7 - Enable Vertical and Horizontal Retrace
D6 - Byte Mode (1), Word Mode (0)
D5 - Address Wrap
D4 - Reserved

D3 - Count by 2
D2 - Multiply Vertical by 2 (use half in CRTC 8,10h,12h,15h,18h)
D1 - Select Row Scan Counter (not used)
D0 - Compatibility Mode Support (enable interleave)

### Index 18h - Line Compare Register

Used in combination with the Start Address register, the Line Compare register provides hardware support for a split screen display. When the horizontal scan counter (total number of horizontal scans) equals the value of the Line Compare register, the display refresh memory address counter will be cleared. This has the effect of breaking the display screen into two separate windows. The upper window on the display screen displays the data that is pointed to by the Start Address register; the lower window on the display screen displays the data that starts at location zero in display memory. The upper window may be scrolled using the Start Address register while the lower window remains stationary.

Line Compare is a 10-bit register. The ninth bit (D8) is located in the Overflow register and the tenth bit (D9) is located in the Max Scan Line register.

## Sequencer Registers

The Sequencer controls the overall timing of all VGA functions, and also performs some memory address decoding. It is controlled through five I/O registers which are multiplexed into two I/O addresses. The Sequencer Index register is mapped at I/O address 3C4, and the Sequencer Data register is mapped at I/O address 3C5. Table 3-6 lists the registers of the Sequencer.

**Table 3-6. Sequencer registers**

| Index | Register |
|---|---|
| 0 | Reset Register |
| 1 | Clock Mode |
| 2 | Color Plane Write Enable |
| 3 | Character Generator Select |
| 4 | Memory Mode |

### Index 0 - Reset Register

D7-D2 - Reserved
D1 - Synchronous Reset
D0 - Asynchronous Reset

**Asynchronous Reset**, when set to zero, will immediately halt and reset the sequencer. This can cause data loss in the display RAM if it is interrupted in midcycle. **Synchronous Reset**, when set to zero, will halt and reset the sequencer at the end of its current cycle.

## Index 1 - Clock Mode Register

D7,D6 - Reserved
D5 - Display Off
D4 - Allow 32-Bit Fetch (not used in standard modes)
D3 - Divide Dot Clock by 2 (used in some 320x200 modes)
D2 - Allow 16-Bit Fetch (used in mono graphics modes)
D1 - Reserved
D0 - Enable (0) 9 Dot Characters (mono text and 400-line text modes)

**Display Off** will blank the screen and give the CPU uninterrupted access the display memory. BIOS service 12h (AH = 12h, BL = 36h) can be used to change this bit.

## Index 2 - Color Plane Write Enable Register

D7,D6 - Reserved
D3 - Plane 3 Write Enable
D2 - Plane 2 Write Enable
D1 - Plane 1 Write Enable
D0 - Plane 0 Write Enable

## Index 3 - Character Generator Select Register

D7,D6 - Reserved
D5 - Character Generator Table Select A (MSB)
D4 - Character Generator Table Select B (MSB)
D3,D2 - Character Generator Table Select A
D1,D0 - Character Generator Table Select B

This register is only of interest if your software will be using multiple character sets. Either one or two character sets can be active. **Character Generator Table Select A** selects which character set will be used for a character whose attribute byte has bit D3 set to zero. **Character Generator Table Select B** selects which character generator will be used for a character whose attribute byte has bit D3 set to one. The value of this register can be changed using BIOS function 11h (SH = 11h, AL = 03h).

## Index 4 - Memory Mode Register

D4 to D7 - Reserved
D3 - Chain 4 (address bits 0&1 to select plane, mode 13h)
D2 - Odd/Even (address bit 0 to select plane 0&2 or 1&3, text modes)
D1 - Extended Memory (disable 64K modes)
D0 - Reserved

# Graphics Controller Registers

The Graphics Controller resides in the data path between display memory and the system processor. In its default state, the Graphics Controller is transparent and data passes directly between processor and display memory. In other configurations, the Graphics Controller can provide a hardware assist to graphics drawing algorithms by performing logical operations on data being written or read by the processor.

Nine Graphics Controller registers are multiplexed into two I/O addresses; address 3CE is the index register and address 3CF is the data register. Table 3-7 lists the registers of the Graphics Controller.

A color plane must be write enabled via the Sequencer Color Plane Write Enable register before any drawing operations can occur in that plane.

**Table 3-7. Graphics Controller registers**

| Index | Register |
|---|---|
| 0 | Set/Reset Register |
| 1 | Set/Reset Enable Register |
| 2 | Color Compare Register |
| 3 | Data Rotate & Function Select |
| 4 | Read Plane Select Register |
| 5 | Mode Register |
| 6 | Miscellaneous Register |
| 7 | Color Don't Care Register |
| 8 | Bit Mask Register |

## Index 0 - Set/Reset Register

D7-D4 - Reserved (0)
D3 - Fill Data for Plane 3
D2 - Fill Data for Plane 2
D1 - Fill Data for Plane 1
D0 - Fill Data for Plane 0

A better name for this register would be Color Fill Data. It is used to define a fill color to be written to display memory during any display memory write operation when Set/Reset mode is enabled (the write data from the processor will be ignored). Set/Reset mode is enabled for each plane individually through the Set/Reset Enable register (Index 1 below).

In 16-color graphics modes a single byte written to display memory defines eight pixels in one or more planes (unless a pixel mask function is enabled). In Set/Reset mode, all eight pixels of each plane will be filled with the fill data for that plane from the Set/Reset register. The write mode must be set to zero (see Mode Register - Index 5).

Individual memory bits may be write protected from a Set/Reset fill operation using the Bit Mask register (Index 8). Other logical functions (such as Rotate, And, Or, or Xor) have no effect on Set/Reset operations. Planes that are not enabled for Set/Reset are under normal control of the other logical functions.

The Set/Reset register can be used to quickly fill regions of the display with a predefined color.

## Index 1 - Set/Reset Enable Register

D7-D4 - Reserved (0)
D3 - Enable Set/Reset for Plane 3 (1 = enable)
D2 - Enable Set/Reset for Plane 2
D1 - Enable Set/Reset for Plane 1
D0 - Enable Set/Reset for Plane 0

Set/Reset Enable defines which memory planes will receive fill data from the Set/Reset register. Any plane that is disabled for Set/Reset will be written with normal processor output data.

## Index 2 - Color Compare Register

D7-D4 - Reserved
D3 - Color Compare Value for Plane 3
D2 - Color Compare Value for Plane 2
D1 - Color Compare Value for Plane 1
D0 - Color Compare Value for Plane 0

The Color Compare register can be used to implement graphics drawing algorithms that must find and identify objects in display memory by their color. Color Compare allows a single display memory read cycle to compare the data of all four planes to a reference color and report whether a color match was found for each pixel position.

For each pixel position, one indicates that the color data in all four planes matched the compare data.

The Color Compare function is enabled through the Mode register.

## Index 3 - Data Rotate/Function Select Register

D7-D5 - Reserved (0)
D4,D3 - Function Select
D2-D0 - Rotate Count

This register controls two independent functions: write data rotation, and logical functions performed on write data.

Data can be rotated during a write cycle for zero- to seven-bit positions. This function can be used to provide hardware support for BITBLT where source and destination are not byte aligned. Write mode 0 must be selected to enable rotation.

Each time a display memory read cycle is performed by the host processor, the read data is latched into a set of on-board latches called the *processor latches*. **Function select** allows write data from the processor to be combined logically with the data stored in these latches. Write mode 0 or 2 must be selected to enable logical functions. Functions are:

| D4 D3 | Function |
|---|---|
| 0 0 | Write data unmodified |
| 0 1 | Write data ANDed with processor latches |
| 1 0 | Write data ORed with processor latches |
| 1 1 | Write data XORed with processor latches |

If both rotation and a logical function are enabled, the rotation occurs before the logical function is applied.

## Index 4 - Read Plane Select Register

D7-D2 - Reserved (0)
D1,D0 - Defines Color Plane for Reading (0-3)

The Read Plane Select register determines which color plane is enabled for reading by the processor (except in Color Compare mode).

## Index 5 - Mode Register

D7 - Reserved (0)
D6 - 256-Color Mode

D5 - Shift Register Mode
D4 - Odd/Even Mode
D3 - Color Compare Mode Enable (1 = enable)
D2 - Reserved (0)
D1,D0 - Write Mode

Most of the bits of the Mode register should not be modified by software. Two fields that are of interest, however, are the Write Mode field, which can be used to control how processor data is written into display memory, and the Color Compare Mode Enable (see Color Compare register).

| D1 D0 | Write Mode |
|---|---|
| 0 0 | Direct write (Data Rotate, Set/Reset may apply) |
| 0 1 | Use processor latches as write data |
| 1 0 | Color plane n (0-3) is filled with the value of bit n in the write data |
| 1 1 | Use (rotated) write data ANDed with Bit Mask as Bit Mask<br>Use Set/Reset as if Set/Reset was enabled for all planes |

Write mode 3 is a new mode for VGA (it is not present in EGA). The most common use for this write mode is during write operations in 16-color graphics modes when only one pixel is being changed. On EGA, similar functions are normally done with one 16-bit I/O instruction to set the Bit Mask Register (index 8) in the Graphics Controller.

```
        ...Initialize offset and mask
        ...Select write mode 0
        ...Enable Set/Reset (select color)
Loop:
        MOV     AH,Mask         ;Fetch pixel mask
        MOV     AL,8            ;Fetch Bit Mask register index
        MOV     DX,3CEh         ;Fetch address of Graphics Controller
        OUT     DX,AX           ;Load Bit Mask register
        OR      ES:[DI],AL      ;Set next pixel
        ...Update pixel offset DI and mask
```

A faster operation results when write mode 3 is used as in following code.

```
        ...Initial offset and mask
        ...Select write mode 3
        ...Select read mode color compare
        ...Set color don't care to 0
        ...Enable Set/Reset (select color)
        MOV     AL,ES:[DI]      ;Set latches to FFh (using color compare)
Loop:
        MOV     AL,Mask         ;Fetch pixel mask
        OR      ES:[DI],AL      ;Set next pixel
        ...Update pixel offset DI and mask
```

Notice that in the second method the I/O instruction in the loop is eliminated, which can result in code which is 50% faster than when write mode 0 is used.

## Index 6 - Miscellaneous Register

D7 to D4 - Reserved
D3 to D2 - Memory Map
0 0 = A000h for 128K
0 1 = A000h for 64K
1 0 = B000h for 32K
1 1 = B800h for 32K
D1 - Odd/Even Enable (used in text modes)
D0 - Graphics Mode Enable

**Memory Map** defines the location and size of the host window.

## Index 7 - Color Don't Care Register

D7-D4 - Reserved (0)
D3 - Plane 3 Don't Care
D2 - Plane 2 Don't Care
D1 - Plane 1 Don't Care
D0 - Plane 0 Don't Care

Color Don't Care is used in conjunction with Color Compare mode. This register masks particular planes from being tested during color compare cycles.

## Index 8 - Bit Mask Register

D7 - Mask Data Bit 7
D6 - Mask Data Bit 6
D5 - Mask Data Bit 5
D4 - Mask Data Bit 4
D3 - Mask Data Bit 3
D2 - Mask Data Bit 2
D1 - Mask Data Bit 1
D0 - Mask Data Bit 0

The Bit Mask register is used to mask certain bit positions from being modified during read-modify-write cycles. It must be noted, however, that the Bit Mask register does

not implement a true bit mask and it must be used very carefully to achieve the desired results.

A zero value in a particular bit of the bit mask register means that during a processor write to display memory, the data for that bit position will be taken from the processor latches rather than from the processor output data. For this to function as a mask operation, the processor latches must be properly loaded through a read operation before a write operation is performed.

# Attribute Controller and Video DAC Registers

The Attribute Controller consists of twenty registers that are multiplexed into one I/O address. The index register and data register are both mapped at address 3C0h, with write cycles alternating between the two. An internal flip-flop toggles with each write operation, selecting the index and data registers alternately. This flip-flop can be initialized by performing and I/O read operation at address 3BA (in monochrome mode) or 3DA (in color mode). After initialization, the first write cycle at address 3C0 will be directed to the index register.

Attribute Controller outputs drive the Video DACs (Digital to Analog Converters), which convert binary color information into analog voltages to drive the display. The Video DAC circuit also includes an additional color lookup table.

## Attribute Controller Registers

### *Index Register*

D7,D6 - Reserved
D5 - Palette Address Source
    0 = palette can be modified, screen is blanked
    1 = screen is enabled, palette cannot be modified
D4-D0 - Palette Register Address

**Palette Register Address** selects which register of the Attribute Controller will be addressed by the next I/O write cycle, as shown in Table 3-8.

**Palette Address Source** selects whether the palette is addressed by display refresh data or by the index register. If set to one, display refresh will occur but palette registers cannot be modified. If set to zero, the palette registers can be programmed but the display will be blanked.

Table 3-8. Attribute Controller registers

| Index | Register |
|---|---|
| 00 | Color Palette register 0 |
| 01 | Color Palette register 1 |
| 02 | Color Palette register 2 |
| 03 | Color Palette register 3 |
| 04 | Color Palette register 4 |
| 05 | Color Palette register 5 |
| 06 | Color Palette register 6 |
| 07 | Color Palette register 7 |
| 08 | Color Palette register 8 |
| 09 | Color Palette register 9 |
| 0A | Color Palette register 10 |
| 0B | Color Palette register 11 |
| 0C | Color Palette register 12 |
| 0D | Color Palette register 13 |
| 0E | Color Palette register 14 |
| 0F | Color Palette register 15 |
| 10 | Mode Control register |
| 11 | Screen Border Color |
| 12 | Color Plane Enable register |
| 13 | Horizontal Panning register |
| 14 | Color Select register |

## Index 00 to 0Fh - The Palette Registers

D6,D7 - Reserved
D0 to D5 - Color Value

Palette registers allow an application program to choose which colors will be displayed at any time. It is not used in 256 color modes.

## Index 10h - Mode Control Register

D7 - P4,P5 Source Select
D6 - Pixel Width
D5 - Horizontal Panning Compatibility
D4 - Reserved
D3 - Background Intensify/Enable Blink
D2 - Line Graphics Enable (text modes only)
D1 - Display Type
D0 - Graphics/Text Mode

**P4,P5 Source Select** selects the source for video outputs P4 and P5 to the DACs. If set to zero, P4 and P5 are driven from the Palette registers (normal operation). If set to one, P4 and P5 video outputs come from bits 0 and 1 of the Color Select register.

**Pixel Width** is set to one for mode 13 (256-color graphics).

**Horizontal Panning Compatibility** enhances the operation of the Line Compare register of the CRT Controller, which allows one section of the screen to be scrolled while another section remains stationary. When this bit is set to one, the stationary section of the screen will also be immune to horizontal panning.

**Background Intensify/Enable Blink** selects which of these two attributes will be enabled by character attribute bit 7 in text modes. If this bit is set to zero, the Background Intensify attribute will be enabled. If this bit is set to one, the Blinking attribute will be enabled.

**Line Graphics Enable** forces, in nine-bit modes (mono text and 400-line text), ninth bit of characters C0h to DFh to match the eighth bit.

**Display Type** determines whether monochrome or color attributes are generated. A zero selects color attributes, one selects monochrome.

**Graphics/Text Mode** determines whether attributes are decoded as four-bit graphics pixels or as byte-wide text attributes. A zero enables text attributes, one enables graphics attributes.

## Index 11h - Screen Border Color

In text modes, the Screen Border Color register selects the color of the border that surrounds the text display area on the screen. This is also referred to by IBM as Overscan. Unfortunately, this feature does not work properly on EGA displays in 350-line modes.

## Index 12h - Color Plane Enable Register

D7,D6 - Reserved
D5,D4 - Video Status Mux
D3 - Enable Color Plane 3
D2 - Enable Color Plane 2
D1 - Enable Color Plane 1
D0 - Enable Color Plane 0

The **Video Status Mux** bits can be used in conjunction with the Diagnostic bits of Input Status register 1 to read palette registers. For the EGA, this is the only means available for reading the palette registers.

**Enable Color Planes** can be used to enable or disable color planes at the input to the color lookup table. A zero in any of these bit positions will mask the data from that

color plane. The effect on the display will be the same as if that color plane were cleared to all zeros.

## Index 13 - Horizontal Panning Register

D7-D4 - Reserved
D3-D0 - Horizontal Pan

**Horizontal Pan** allows the display to be shifted horizontally one pixel at a time. Values are interpreted according to mode selected as shown in Table 3-9.

Table 3-8. Attribute Controller registers

| Value | Number of pixels shifted to the left | | |
|---|---|---|---|
| | 0+, 1+, 2+ 3+, 7, 7+ | 13h | Other modes |
| 0 | 1 | 0 | 0 |
| 1 | 2 | 1 | |
| 2 | 3 | 2 | 1 |
| 3 | 4 | 3 | |
| 4 | 5 | 4 | 2 |
| 5 | 6 | 5 | |
| 6 | 7 | 6 | 3 |
| 7 | 8 | 7 | |
| 8 | 9 | | |

## Index 14 - Color Select Register

D7-D4 - Reserved
D3 - Color 7
D2 - Color 6
D1 - Color 5
D0 - Color 4

**Color 7 and Color 6** are normally used as the high order bits of the eight-bit video color data from the attribute controller to the video DACs. The only exceptions are 256-color modes.

**Color 5 and Color 4** can be used in place of the P5 and P4 outputs from the palette registers (see Mode Control Register - Index 10).

In 16-color modes, the color select register can be used to rapidly cycle between sets of colors in the video DAC.

## Video DAC Registers (I/O Addresses 3C6, 3C7, 3C8, and 3C9)

The VGA video DAC is actually three video DACs (one each for red, green, and blue), preceded by a color lookup table. Each video DAC converts six bits of binary color information into an analog voltage for driving the display. The color lookup table converts the eight bits that are output from the VGA Attribute Controller into eighteen bits (six for each video DAC). This gives the VGA the capability of displaying 256 simultaneous colors from a palette of 262, 144.

Five registers are used to access the video DAC:

3C6 - Pixel Mask register
3C7 - DAC State register (Read-only)
3C7 - Lookup Table Read Index register (Write-only)
3C8 - Lookup Table Write Index register
3C9 - Lookup Table Data register

Two separate index registers are used for selecting among the 256 internal color registers of the lookup table. The Read Index is used only when data is read from the lookup table, and the Write Index is used only when data is being written to the lookup table. A color register, which is eighteen bits wide, is programmed by writing an eight bit index to the **Lookup Table Write Index register (3C8)**, then writing three six-bit values to the **Lookup Table Data register (3C9)**. The index register will automatically increment after the third byte is written, so that a block of color registers can be programmed without repeatedly setting the index.

A color register can be read by writing an eight-bit index into the **Lookup Table Read Index register (3C7)**, then reading three six-bit values from the **Lookup Table Data register (3C9)**. The index register will automatically increment after the third byte is read.

The **DAC State register (3C7)** can be used to determine whether the color lookup table is currently configured for a register read operation or a register write operation. A value of zero in bits D0 and D1 indicates that the lookup table is in a write mode.

Unlike the Attribute Controller, processor accesses to the color lookup table in the DAC can be performed at any time; on some VGAs, however, these accesses may interfere with screen refresh and cause 'snow' on the display. This can be avoided by waiting for vertical retrace before programming it.

# 4

# *The ROM BIOS*

# What is the ROM BIOS?

The VGA ROM BIOS is a set of low level firmware routines that are accessed by executing a software interrupt instruction (INT 10H) with parameters specified in registers.

# Individual BIOS Functions

## Function 0: Mode Select

**Input Parameters:**

AH = 0
AL = Mode number (0 to 13H)

If AL bit D7 equals 0, the display buffer will be cleared. If bit D7 equals 1, the display buffer will be left unmodified.

**Return Value:** None.

## Function 1: Set Cursor Size

This function defines cursor height.

**Input Parameters:**

AH = 1
CH = start scan line (0 - 31)
CL = end scan line (0 - 31)

**Return Value:** None.

## Function 2: Set Cursor Position

This function will position the cursor at a specified location on the display screen. A separate cursor is maintained for each display page.

**Input Parameters:**

AH = 2
BH = display page number
DH = Row
DL = Column

**Return Value:** None.

## Function 3: Read Cursor Size and Position

This function returns data on the cursor position and cursor height.

**Input Parameters:**

AH = 3
BH = Display page number

**Return Value:**

CH = cursor start scan line
CL = cursor end scan line
DH = cursor row
DL = cursor column

## Function 4: No Standard Support (Get Light Pen)

## Function 5: Select Active Page

This function selects which display page is displayed on the screen.

**Input Parameters:**

AH = 5
AL = display page number

**Return Value:** None.

## Function 6: Scroll Text Window Up (or Blank Window)

This function scrolls a specified portion of the display (the scroll window) upward.

**Input Parameters:**

AH = 6
AL = number of lines to scroll
    (AL = 0 blanks window to all spaces)
BH = text attribute to use when filling blank lines at bottom of window
CH = row number of upper left corner of window
CL = column number of upper left corner of window
DH = row of lower right corner of window

DL = column of lower right corner of window

**Return Value:** None.

## Function 7: Scroll Text Window Down (or Blank Window)

This function scrolls a specified portion of the display (the scroll window) downward.

**Input Parameters:**

AH = 7
AL = number of lines to scroll
(AL = 0 blanks window to all spaces)
BH = text attribute to use when filling blank lines at top of window
CH = row number of upper left corner of window
CL = column number of upper left corner of window
DH = row of lower right corner of window
DL = column of lower right corner of window

**Return Value:** None.

## Function 8: Read Character and Attribute at Cursor Position

**Input Parameters:**

AH = 8
BH = display page number

**Return Value:**

AL = character code
AH = character attribute (text modes only)

## Function 9: Write Character and Attribute at Cursor Position

**Input Parameters:**

AH = 9
AL = character code
BH = display page number
BL = attribute (text modes) or color value (graphics modes)
CX = repetition count (up to end of current row)

**Return Value:** None.

## Function 0Ah: Write Character Only at Cursor Position

This function writes an ASCII character to display memory at the current cursor position. The previous attribute is preserved. The cursor position is not incremented.

**Input Parameters:**

AH = 0Ah
AL = character code
BH = display page number
BL = color value (graphics modes)
CX = repetition count (up to end of current row)

If the VGA is operating in a graphics mode and bit D7 of register BL equals 1, the character being written will be exclusive ORed, XORed, with the previous data in display memory.

**Return Value:** None.

## Function 0Bh: Set CGA Color Palette (Modes 4,5,6)

This function configures the VGA to emulate one of the two standard CGA graphics color palettes.

**Input Parameters:**

AH = 0Bh
If BH = 0:
    BL = graphics background color or text border color
If BH = 1:
    BL = palette number (0 or 1)

**Return Value:** None.

## Function 0Ch: Write Graphics Pixel

This function is a slow method for manipulating pixels in graphics mode.

**Input Parameters:**

AH = 0Ch
AL = pixel value

CX = pixel column number
DX = pixel row number

If bit D7 of register AL is set to one, the new pixel value will be exclusive ORed, XORed, with the existing background color.

**Return Value**: None.

## Function 0Dh: Read Graphics Pixel

**Input Parameters**:

AH = 0Dh
CX = pixel column number
DX = pixel row number

**Return Value**:

AL = pixel value

## Function 0Eh: Write Character and Advance Cursor

The character is displayed at the current cursor position, and the cursor is automatically advanced to the next character position. At the end of a line, the cursor will wrap around to the next line. ASCII BELL, BACKSPACE, CARRIAGE RETURN and LINEFEED are recognized and their functions are performed accordingly. Vertical scrolling is performed as required.

If the VGA is operating in a text mode, the character attribute is left unmodified. If the VGA is operating in a graphics mode, the character color may be specified in the call.

Function 0Eh is used by the standard MS-DOS console driver for screen handling.

**Input Parameters**:

AH = 0Eh
AL = character code
BL = character color (graphics modes only)

**Return Value**: None.

## Function 0Fh: Get Current Display Mode

**Input Parameters:**

AH = 0Fh

**Return Value:**

AH = number of display columns
AL = display mode
BH = active display page

## Function 10h: Set EGA Palette Registers

This function is divided into 14 subfunctions that control color translations.

### *Subfunction 0: Program a Palette Register*

**Input Parameters:**

AH = 10h
AL = 00h
BL = palette register number (0 to Fh)
BH = color data (0 to 3Fh)

**Return Value:** None.

### *Subfunction 1: Set Border Color (Overscan)*

**Input Parameters:**

AH = 10h
AL = 01h
BH = color data (0 to FFh)

**Return Value:** None.

### *Subfunction 2: Set All Palette Registers*

**Input Parameters:**

AH = 10h
AL = 02h
ES:DX = address of 17-byte buffer (16 palette values plus overscan value)

**Return Value:** None.

## Subfunction 3: Blink/Intensity Attribute Control

This subfunction provides a convenient method of toggling the control bit that defines whether the blinking attribute is enabled or the intensified background attribute is enabled.

**Input Parameters:**

AH = 10h
AL = 03h
BL = 0 - enable background intensify
BL = 1 - enable foreground blink

**Return Value:** None.

## Subfunction 7: Read a Single Palette Register

**Input Parameters:**

AH = 10h
AL = 7
BL = register number (0-15)

**Return Value:**

BH = palette register value

## Subfunction 8: Read Border Color (Overscan) Register

**Input Parameters:**

AH = 10h
AL = 8

**Return Value:**

BH = Border Color Register value

## Subfunction 9: Read All Palette Registers

**Input Parameters:**

AH = 10h
AL = 9

ES:DX = address of 17-byte buffer (16 palette values plus overscan value)

**Return Value:**

17 bytes stored at [ES:DX]

### *Subfunction 10h: Set a Single DAC Register*

This subfunction sets the 18-bit color value in a single DAC register.

**Input Parameters:**

AH = 10h
AL = 10h
BX = DAC register number (0 to FFh)
DH = Red intensity level (0 to 3Fh)
CH = Green intensity level (0 to 3Fh)
CL = Blue intensity level (0 to 3Fh)

**Return Value:** None.

### *Subfunction 12h: Set Block of DAC Registers*

This subfunction sets the 18-bit color values in a block of DAC registers.

**Input Parameters:**

AH = 10h
AL = 12h
BX = starting DAC register (0 to 255)
CX = number of registers to set (1 to 256)
ES:DX = address of color table

The color table consists of 3 bytes per register (red, green, and blue, each in range 0 to 3Fh).

**Return Value:** None.

### *Subfunction 13h: Select Color Subset*

This subfunction selects one of up to 16 color subsets.

**Input Parameters:**

AH = 10h
AL = 13h

If BL = 0: Select mode
BH = 0: 4 subsets of 64 colors
BH = 1: 16 subsets of 16 colors
If BL = 1: Select subset
BH = subset (0-16)

**Return Value**: None.

## *Subfunction 15h: Read a Single DAC Register*

**Input Parameters**:

AH = 10h
AL = 15h
BX = DAC register number (0-255)

**Return Value**:

DH = red intensity level (0 to 3Fh)
CH = green intensity level (0 to 3Fh)
CL = blue intensity level (0 to 3Fh)

## *Subfunction 17h: Read Block of DAC Registers*

**Input Parameters**:

AH = 10h
AL = 17h
BX = starting DAC register number (0-255)
CX = number of registers (1-256)
ES:DX = destination address for register data

**Return Value**:

Register data at destination address (3 bytes per register)

## *Subfunction 18h: Set PEL Mask*

**Input Parameters**:

AH = 10h
AL = 18h
BL = PEL Mask

**Return Value**: None.

### Subfunction 19h: Read PEL Mask

**Input Parameters:**

AH = 10h
AL = 19h

**Return Value:**

BL = PEL Mask

### Subfunction 1Ah: Read Subset Status

This subfunction returns the number of the current color subset.

**Input Parameters:**

AH = 10h
AL = 1Ah

**Return Value:**

BH = number of current color subset
BL = 0 if 4 subsets are available
BL = 1 if 16 subsets are available

### Subfunction 1Bh: Convert DAC Registers to Gray Scale

This subfunction converts a block of DAC registers from color values to monochrome gray scale values, using the following formula:

$$gray = 30\%\ Red + 59\%\ Green + 11\%\ Blue$$

**Input Parameters:**

AH = 10h
AL = 1bh
BX = starting DAC register number (0-255)
CX = number of registers (1-256)

**Return Value:** None.

## Function 11h: Load Character Generator

Function 11h consists of 17 subfunctions which are used to control appearance of text.

### Subfunction 0: Load Custom Character Generator

**Input Parameters:**

AH = 11h
AL = 0
ES:BP = address of character data in system RAM
CX = number of characters to load (1 to 256)
DX = character offset into character generator table
(0 to 255 - for loading a partial character set)
BL = which character generator to load
BH = number of bytes per character (1 to 32)

**Return Value:** None.

### Subfunction 1: Load 8 x 14 Character Set

**Input Parameters:**

AH = 11h
AL = 1
BL = which character generator to load (0 to 7)

**Return Value:** None.

### Subfunction 2: Load 8 x 8 Character Set

**Input Parameters:**

AH = 11h
AL = 2
BL = which character generator to load (0 to 7)

**Return Value:** none

### Subfunction 3: Select Active Character Set(s)

This subfunction selects which of the VGA's eight character generator tables will be active.

**Input Parameters:**

AH = 11h
AL = 3
BL(D0,D1,D4) - Selects which character generator will be active

for a character with attribute bit 3 = 0
BL(D2,D3,D5) - Selects which character generator will be active
for a character with attribute bit 3 = 1

**Return Value**: None.

### Subfunction 4: Load 8 x 16 Character Set

**Input Parameters**:

AH = 11h
AL = 4
BL = which character generator to load (0-7)

**Return Value**: None.

### Subfunctions 10h, 11h, 12h, 14h

These subfunctions are identical to functions 0, 1, 2 and 4, except that CRTC is reprogrammed to match the selected character size.

### Subfunction 20h: Initialize INT 1Fh Vector (Modes 4-6)

This subfunction initializes the vector that points to characters 80h through FFh of 8x8 font used in graphics modes 4, 5 and 6.

**Input Parameters**:

AH = 11h
AL = 20h
ES:BP = Pointer to character definitions

**Return Value**: None.

### Subfunction 21h: Set Graphics Mode to Display Custom Character Set

In graphics modes, this subfunction sets BIOS variables so that text can be drawn using a custom character set. The character set must remain resident in system memory.

**Input Parameters**:

AH = 11h
AL = 21h
ES:BP = address of custom character table

CX = bytes per character
BL = number of character rows to be displayed:
1 = 14 character rows
2 = 25 character rows
3 = 43 character rows
0 = DL contains number of character rows

**Return Value**: None.

## Subfunction 22h: Set Graphics to Display 8 x 14 Text

In graphics modes, this subfunction will set BIOS variables so that the standard 8x14 character set is used to draw characters.

**Input Parameters**:

AH = 11h
AL = 22H
BL = number of character rows on screen:
1 = 14 character rows
2 = 25 character rows
3 = 43 character rows
0 = DL contains number of character rows

Not all values will result in satisfactory appearance

**Return Value**: None.

## Subfunction 23h: Initialize Graphics Mode to Display 8 x 8 Text

In graphics modes, this subfunction will set BIOS variables so that the standard 8x8 character set is used to draw characters.

**Input Parameters**:

AH = 11h
AL = 23H
BL = number of character rows on screen:
1 = 14 character rows
2 = 25 character rows
3 = 43 character rows
0 = DL contains number of character rows

Not all values will result in satisfactory appearance

**Return Value**: None.

## Subfunction 24h: Initialize Graphics Mode to Display 8 x 16 Text

In graphics modes, this subfunction will set the BIOS variables to use standard 8x16 character set to draw characters.

**Input Parameters**:

AH = 11h
AL = 24H
BL = number of character rows on screen:
    BL = 1 - 14 character rows
    BL = 2 - 25 character rows
    BL = 3 - 43 character rows

**Return Value**: None.

## Subfunction 30h: Return Information About Current Character Set

This subfunction can be used to read information about the current character set being used.

**Input Parameters**:

AH = 11h
AL = 30h
BH = Information type requested
    BH = 0: return current INT 1FH pointer
    BH = 1: return current INT 43H pointer
    BH = 2: return pointer to Enhanced (8x14) character set
    BH = 3: return pointer to CGA (8x8) character set
    BH = 4: return pointer to upper half of CGA 8x8 char set
    BH = 5: return pointer to alternate 9x14 monochrome characters
    BH = 6: return pointer to 8x16 characters
    BH = 7: return pointer to alternate 9x16 characters

**Return Values:**

CL = character height (number of rows in a character)
DL = character rows on screen - 1
ES:BP = return pointer

# Function 12: Get VGA Status (Set Alternate Print Screen)

Function 12h is a group of unrelated functions which share the same function number.

## Subfunction 10h: Return VGA Information

This subfunction returns information on the current VGA configuration.

**Input Parameters:**

AH = 12h
BL = 10h

**Return Values:**

BH = 0 Color mode in effect (3Dx)
1 Mono mode in effect (3Bx)
BL = Memory size: 0 = 64k, 1 = 128k, 2 = 192k, 3 = 256k
CH = Feature bits
CL = EGA switch settings

## Subfunction 20h: Revector Print Screen (INT 05h) Interrupt

**Input Parameters:**

AH = 12h
BL = 20h

**Return Values:** None.

## Subfunction 30h: Select Scan Line Count for Next Text Mode

**Input Parameters:**

AH = 12h
AL = Number of scan lines: 0 = 200, 1 = 350, 2 = 400
Will take effect on next mode select
for modes 0 to 3 and 7.
BL = 30h

**Return Values:**

AL = 12h indicating that function is supported (0 if VGA not active)

### Subfunction 31h: Enable/Disable Palette Load During Mode Set

**Input Parameters:**

AH = 12h
AL = 0 enable (default), 1 disable
BL = 31h

**Return Values:**

AL = 12h indicating that function is supported (0 if VGA not active)

### Subfunction 32h: Enable/Disable VGA Access

**Input Parameters:**

AH = 12h
AL = 0 enable, 1 disable I/O and memory access to VGA
BL = 32h

**Return Values:**

AL = 12h indicating that function was performed (AL was 0 or 1)

### Subfunction 33h: Enable/Disable Gray Scale Summing

**Input Parameters:**

AH = 12h
AL = 0 enable, 1 disable gray scale summing
BL = 33h

**Return Values:**

AL = 12h indicating that function is supported (0 if VGA not active)

### Subfunction 34h: Enable/Disable CGA/MDA Cursor Emulation.

**Input Parameters:**

AH = 12h
AL = 0 enable, 1 disable CGA cursor emulation
BL = 34h

**Return Values:**

AL = 12h indicating that function is supported (AL was 0 or 1)

### *Subfunction 35h: Switch Displays.*

**Input Parameters:**

AH = 12h
AL = Select video:
  0 - Initial adapter video system off (before call with AL = 1)
  1 - Initial motherboard video system on (after call with AL = 0)
  2 - Switch to inactive BIOS and video system (before call with AL = 3)
  3 - Initialize video system with parameters in ES:DX (after call with AL = 0 or 2)
BL = 35h
ES:DX = address of 128-byte save area (for AL = 0, 2, or 3)

**Return Values:**

AL = 12h indicating that function is supported (0 if VGA not active)

### *Subfunction 36h: Display On/Off*

**Input Parameters:**

AH = 12h
AL = 0 enable, 1 disable video output (maximum access to display memory)
BL = 36h

**Return Values:**

AL = 12h indicating that function is supported (0 if VGA not active)

## Function 13h: Write Text String

The text string may be straight character codes (ASCII data), or it may include embedded attribute data. The cursor may be advanced to the end of text, or it may be left unmodified. The ASCII characters for BELL (7), BACKSPACE (8), CARRIAGE RETURN (0D hex) and LINEFEED (0A hex) are recognized and their appropriate functions performed.

**Input Parameters:**

AH = 13h
BH = display page number

CX = character count (length of string)
DH = row for start of string
DL = column for start of string
ES:BP = address of source text string in system RAM
AL = mode:
- 0: BL = Attribute for all characters - Cursor is not updated
- 1: BL = Attribute for all characters - Cursor is updated
- 2: String contains alternating character codes and Attributes - Cursor is not updated
- 3: String contains alternating character codes and Attributes - Cursor is updated

**Return Value**: None.

## Function 1Ah: Read or Write Configuration

This function is divided into two subfunctions that read or modify information on the current configuration of display devices in the system.

### *Subfunction 0: Read Display Configuration Code*

**Input Parameters**:

AH = 1Ah
AL = 0

**Return Values**:

AL = 1Ah
BL = primary display
BH = secondary display

Display information is interpreted as follows:

0 = no display
1 = MDA
2 = CGA
3 = EGA with ECD display
4 = EGA with CD display
5 = EGA with Monochrome Display
6 = PGC (Professional Graphics Controller)
7 = VGA with monochrome display
8 = VGA with color display
0Bh = MCGA with monochrome display
0Ch = MCGA with color display

### Subfunction 1: Write Display Configuration Code

**Input Parameters:**

AH = 1Ah
AL = 1
BL = primary display info
BH = secondary display info

For an explanation of info codes, see subfunction 0.

**Return Value:**

AL = 1Ah

## Function 1Bh: Return VGA Status Information

**Input Parameters:**

AH = 1Bh
BX = 0
ES:DI = pointer to 64 byte buffer for return data

**Return Values:**

AL = 1Bh

The return buffer will contain information as shown in Table 4-1.

**Table 4-1. VGA functionality and video state information**

| Byte Number | Size | Contents |
|---|---|---|
| 0 | dword | Pointer to Static Functionality Table (see table 4-2) |
| 4 | byte | Current display mode |
| 5 | word | Number of character columns |
| 7 | word | Size of video data area (REGEN BUFFER) in bytes |
| 9h | word | Current offset within REGEN BUFFER |
| 0Bh | 8 words | Cursor positions, two words per page, for up to 8 pages |
| 1Bh | byte | Cursor end |
| 1Ch | byte | Cursor start |
| 1Dh | byte | Current display page |
| 1Eh | word | CRT Controller address (3B4h or 3D4h) |
| 20h | byte | CGA/MDA mode register value (value of 3B8h/3D8h) |
| 21h | byte | CGA/MDA color register value (value of 3B9h/3D9h) |
| 22h | byte | Number of text rows |

**Table 4-1. VGA functionality and video state information** *(continued)*

| Byte Number | Size | Contents |
|---|---|---|
| 23h | byte | Character height (in scan lines) |
| 25h | byte | Display Configuration Code (active display) |
| 26h | byte | Display Configuration Code (inactive display) |
| 27h | word | Number of colors in current mode (0 for mono modes) |
| 29h | byte | Number of display pages in current mode |
| 2Ah | byte | Number of scan lines in current mode: 0 = 200, 1 = 350, 2 = 400, 3 = 480 |
| 2Bh | byte | Primary character generator (0-7) |
| 2Ch | byte | Secondary character generator (0-7) |
| 2Dh | byte | Miscellaneous state information: |
| | | D5 = 1 - Blinking enabled |
| | | D5 = 0 - Background intensify enabled |
| | | D4 = 1 - CGA cursor emulation enabled |
| | | D3 = 1 - Default palette initialization disabled |
| | | D2 = 1 - Monochrome display attached |
| | | D1 = 1 - Gray scale conversion enabled |
| | | D0 = 1 - All modes supported on all monitors |
| 2Eh | byte | Reserved |
| 2Fh | byte | Reserved |
| 30h | byte | Reserved |
| 31h | byte | Size of display memory: 0 = 64KB 1 = 128KB 2 = 192KB 3 = 256KB |
| 32h | byte | Save Pointer State Information |
| | | D5 = 1 - DCC extension is active (DCC override) |
| | | D4 = 1 - Palette override active |
| | | D3 = 1 - Graphics font override active |
| | | D2 = 1 - Alpha font override active |
| | | D1 = 1 - Dynamic save area active |
| | | D0 = 1 - 512 Character set active |
| 33h to 33F | | Reserved |

**Table 4-2. VGA static functionality table**

| Byte Number | Size | Contents |
|---|---|---|
| 0 | byte | Video modes supported (1 indicates mode supported):<br>D7 - mode 7<br>D6 - mode 6<br>D5 - mode 5<br>D4 - mode 4<br>D3 - mode 3<br>D2 - mode 2<br>D1 - mode 1<br>D0 - mode 0 |
| 1 | byte | Video modes supported (1 indicates mode supported):<br>D7 - mode 0Fh<br>D6 - mode 0Eh<br>D5 - mode 0Dh<br>D4 - mode 0Ch<br>D3 - mode 0Bh<br>D2 - mode 0Ah<br>D1 - mode 9<br>D0 - mode 8 |
| 2 | byte | Video modes supported (1 indicates mode supported):<br>D7 - Reserved<br>D6 - Reserved<br>D5 - Reserved<br>D4 - Reserved<br>D3 - mode 13h<br>D2 - mode 12h<br>D1 - mode 11h<br>D0 - mode 10h |
| 3 to 6 | | Reserved |
| 7 | byte | Scan line available in text modes (1 indicates supported):<br>D2 - 400 lines<br>D1 - 350 lines<br>D0 - 200 lines |
| 8 | byte | Maximum number of simultaneously displayable character generators |
| 9 | byte | Number of available character generators |
| 0Ah | byte | Miscellaneous BIOS capabilities (1 indicates function supported):<br>D7 - Color paging (fn 10h)<br>D6 - DAC loading (fn 10h)<br>D5 - EGA palette loading (fn 10h)<br>D4 - CGA cursor emulation (fn 1 and 12h)<br>D3 - Palette loading after mode set (fn 0 and 12h)<br>D2 - Character generator loading (fn 11h) |

Table 4-2. VGA static functionality table *(continued)*

| Byte Number | Size | Contents |
|---|---|---|
| | | D1 - Gray scale summing (fn 10h and 12h) |
| | | D0 - All modes on all displays |
| 0Bh | byte | Miscellaneous BIOS capabilities (1 indicates function supported): |
| | | D7 - Reserved |
| | | D6 - Reserved |
| | | D5 - Reserved |
| | | D4 - Reserved |
| | | D3 - DCC (fn 1Ah) |
| | | D2 - Blink/Intensify select (fn 10h) |
| | | D1 - Save/Restore video state (fn 1Ch) |
| | | D0 - Light pen (fn 4) |
| 0Ch to 0Dh | | Reserved |
| 0Eh | byte | Save area function support (1 indicates supported): |
| | | D7 - Reserved |
| | | D6 - Reserved |
| | | D5 - DCC extensions |
| | | D4 - Palette override |
| | | D3 - Text character generator override |
| | | D2 - Graphics character generator override |
| | | D1 - Dynamic save area |
| | | D0 - 512 simultaneous characters |
| 0Fh | | Reserved |

## Function 1Ch: Save/Restore Display Adapter State

This function is divided into three subfunctions that return required buffer size, save display adapter state, and restore display adapter state.

### *Subfunction 0: Return Required Buffer Size*

**Input Parameters:**

AH = 1Ch
AL = 0
CX = Type of data to be saved:
    D0 - Registers
    D1 - BIOS data area
    D2 - DAC registers

**Return Value:**

AL = 1Ch
BX = Required buffer size (in 64 byte blocks)

### *Subfunction 1: Save Display Adapter State*

**Input Parameters:**

AH = 1Ch
AL = 1
CX = Type of data to be saved:
    D0 - Registers
    D1 - BIOS data area
    D2 - DAC registers
ES:BX = Pointer to save buffer

**Return Value:**

AL = 1Ch

### *Subfunction 2: Restore Display Adapter State*

**Input Parameters:**

AH = 1Ch
AL = 2
CX = Type of data to be restored:
    D0 - Registers
    D1 - BIOS data area
    D2 - DAC registers
ES:BX = Pointer to save buffer

**Return Value:**

AL = 1Ch

## The BIOS Data Area

The BIOS data area is a section of the low memory where various BIOS services keep their working variables. Variables used by Video Services are summarized in Table 4-3. Programs which directly alter the status of the display without using the BIOS calls (such as cursor position in CRTC registers) should update these variables to avoid confusing the BIOS.

**Table 4-3. BIOS Data Area**

| Address | Size | Contents | |
|---|---|---|---|
| 0000:0410h | byte | EQUIPMENT_FLAG<br>Bits D4 and D5 of this byte identify the current primary display device:<br>**D5 D4 — Adapter**<br>0 0 — Reserved<br>0 1 — Color 40x25<br>1 0 — Color 80x25<br>1 1 — Monochrome | |
| 0000:0449h | byte | VIDEO_MODE | (current mode) |
| 0000:044Ah | word | COLUMNS | (number of text columns) |
| 0000:044Ch | word | PAGE_LENGTH | (length of each page in bytes) |
| 0000:044Eh | word | START_ADDR | (Start Address register value) |
| 0000:0450h | 8 words | CURSOR_POSITION | (cursor positions for all pages) |
| 0000:0460h | word | CURSOR_SHAPE | (Cursor Start and End registers) |
| 0000:0462h | byte | ACTIVE_PAGE | (current active page number) |
| 0000:0463h | word | CRTC_ADDRESS | (3B4h or 3D4h) |
| 0000:0465h | byte | MODE_REG_DATA | (CGA Mode register setting) |
| 0000:0466h | byte | PALETTE | (CGA Color register setting) |
| 0000:0484h | byte | ROWS | (number of text rows - 1) |
| 0000:0485h | word | CHAR_HEIGHT | (bytes per char) |
| 0000:0487h | byte | EGA_INFO_1<br>D7 = bit D7 from AL on most recent mode select<br>(1 indicates memory was not cleared by mode select)<br>D6,D5 = Display memory size (00=64K, 01=128K, 10=192K, 11=256K)<br>D4 = reserved<br>D3 = 0 indicates VGA is the primary display<br>D2 = 1 will force the BIOS to wait for Vertical Retrace before writing to display memory<br>D1 - 1 indicates that VGA is in monochrome mode<br>D0 - 0 means that CGA cursor emulation is enabled | |
| 0000:0488h | byte | EGA_INFO_2<br>D4-D7 = Feature connector settings<br>D0-D3 = Switch settings | |
| 0000:0489h | byte | MISC_FLAGS | D7&D4 = Scanline count:<br>0 0 = 350 lines<br>0 1 = 400 lines<br>1 0 = 200 lines<br>1 1 = reserved<br>D6 = Display switching enabled |

Table 4-3. BIOS Data Area *(continued)*

| Address | Size | Contents | |
|---|---|---|---|
| | | | D3 = Default palette loading disabled |
| | | | D2 = Monochrome monitor |
| | | | D1 = Gray scale summing enabled |
| | | | D0 = All modes on all displays |
| 0000:048Ah | byte | DCC_INDEX | Index of current video combination |
| 0000:04A8h | dword | SAVE_AREA_PTR | Pointer to save area (see Table 4-4) |

Table 4-4. VGA BIOS save area

| Byte Number | Size | Contents |
|---|---|---|
| 0 | dword | Mandatory pointer to **Video Parameter Table** (see Table 4-5) |
| 4 | dword | Optional pointer to Dynamic Save Area. (This 256-byte table contains 16 palette register values and Overscan register value.) |
| 8 | dword | Optional pointer to **Text Mode Auxiliary Character Set** (see Table 4-6) |
| 0Ch | dword | Optional pointer to **Graphics Mode Auxiliary Character Set** (see Table 4-7) |
| 10h | dword | Optional pointer to **Secondary Save Area** (see Table 4-8) |
| 14h | dword | Reserved |
| 18h | dword | Reserved |

Note: At system initialization, the Environment Pointer is set to point to an Environment Table in ROM. This default Environment Table has only one entry (the Video Parameter Table Pointer). To modify the Environment Table, first copy it from ROM to RAM and then update the Environment Pointer.

Table 4-5. VGA BIOS Video Parameter Table

| Byte Number | Contents | |
|---|---|---|
| 0 | Number of text columns | |
| 1 | Number of text rows minus one | |
| 2 | Character height (in pixels) | |
| 3 and 4 | Display page length (in bytes) | |
| | Sequencer register values: | |
| 5 | | Clock Mode register |
| 6 | | Color Plane Write Enable register |
| 7 | | Character Generator Select register |
| 8 | | Memory Mode register |
| 9 | Miscellaneous register | |
| | CRT Controller register values: | |
| 0ah | | Horizontal Total register |
| 0bh | | Horizontal Display End register |
| 0ch | | Start Horizontal Blanking register |

Table 4-5. VGA BIOS Video Parameter Table *(continued)*

| Byte Number | Contents | |
|---|---|---|
| 0dh | | End Horizontal Blanking register |
| 0eh | | Start Horizontal Retrace register |
| 0fh | | End Horizontal Retrace register |
| 10h | | Vertical Total register |
| 11h | | Overflow register |
| 12h | | Preset Row Scan register |
| 13h | | Maximum Scan Line register |
| 14h | | Cursor Start |
| 15h | | Cursor End |
| 16h-19h | | Unused |
| 1ah | | Vertical Retrace Start register |
| 1bh | | Vertical Retrace End register |
| 1ch | | Vertical Display End register |
| 1dh | | Offset register |
| 1eh | | Underline Location register |
| 1fh | | Start Vertical Blanking register |
| 20h | | End Vertical Blanking register |
| 21h | | Mode Control register |
| 22h | | Line Compare register |
| | Attribute Controller register values: | |
| 23h | | Palette register 0 |
| 24h | | Palette register 1 |
| 25h | | Palette register 2 |
| 26h | | Palette register 3 |
| 27h | | Palette register 4 |
| 28h | | Palette register 5 |
| 29h | | Palette register 6 |
| 2ah | | Palette register 7 |
| 2bh | | Palette register 8 |
| 2ch | | Palette register 9 |
| 2dh | | Palette register 10 |
| 2eh | | Palette register 11 |
| 2fh | | Palette register 12 |
| 30h | | Palette register 13 |
| 31h | | Palette register 14 |
| 32h | | Palette register 15 |
| 33h | | Mode Control register |
| 34h | | Screen Border Color (Overscan) register |
| 35h | | Color Plane Enable register |
| 36h | | Horizontal Panning register |
| | Graphics Controller register values: | |
| 37h | | Set/Reset register |

**Table 4-5. VGA BIOS Video Parameter Table** *(continued)*

| Byte Number | Contents |
|---|---|
| 38h | Set/Reset Enable register |
| 39h | Color Compare register |
| 3ah | Data Rotate & Function Select register |
| 3bh | Read Plane Select register |
| 3ch | Mode register |
| 3dh | Miscellaneous register |
| 3eh | Color Don't Care register |
| 3fh | Bit Mask register |

Modes are ordered in the parameter table as follows:

| Table | Mode |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| 10 | A |
| 11 | B |
| 12 | C |
| 13 | D |
| 14 | E |
| 15 | F (64K display RAM) |
| 16 | 10 (64K display RAM) |
| 17 | F (more than 64K) |
| 18 | 10 (more than 64K) |
| 19 | 0* |
| 20 | 1* |
| 21 | 2* |
| 22 | 3* |
| 23 | 0+, 1+ |
| 24 | 2+, 3+ |
| 25 | 7+ |
| 26 | 11 |
| 27 | 12 |
| 28 | 13 |

**Table 4-6. VGA BIOS Text Mode Auxiliary Character Set Table**

| Byte Number | Size | Contents |
|---|---|---|
| 0 | byte | Bytes per character |
| 1 | byte | Character Map # (0-3 for EGA, 0-7 for VGA) |
| 2 | word | # of characters |
| 4 | word | First character # |
| 6 | dword | Pointer to character set in system memory |
| 10 | byte | Character height (in pixels) |
| 11-n | bytes | List of modes this character set is compatible with, terminated by FFh |

**Table 4-7. VGA BIOS Graphics Mode Auxiliary Character Set Table**

| Byte Number | Size | Contents |
|---|---|---|
| 0 | byte | Number of character rows on display |
| 1 | word | Bytes per character |
| 3 | dword | Pointer to character set in system memory |
| 7-n | bytes | List of modes this character set is compatible with, terminated by FFh |

**Table 4-8. VGA BIOS Secondary Save Area Table**

| Byte Number | Size | Contents |
|---|---|---|
| 0 | word | Length of this table |
| 2 | dword | Pointer to DCC table (see Table 4-9) |
| 6 | dword | Pointer to second Text Mode Auxiliary Character Set (see Table 4-6) |
| 0Ah | dword | Pointer to User Palette Table (see Table 4-10) |
| 0Eh | dword | Reserved |
| 12h | dword | Reserved |
| 16h | dword | Reserved |

**Table 4-9. VGA BIOS Device Combination Code Table**

| Byte Number | Size | Contents |
|---|---|---|
| 0 | byte | Number of entries in this table |
| 1 | byte | Version number |
| 2 | byte | Maximum display type code |
| 3 | byte | Reserved |
| 4 - n | words | List of valid video combinations, one pair per combination<br>Pairs are built from the following values:<br>0 = no display<br>1 = MDA<br>2 = CGA<br>3 = Reserved<br>4 = EGA with CD or ECD display<br>5 = EGA with Monochrome Display<br>6 = PGC (Professional Graphics Controller)<br>7 = VGA with monochrome display<br>8 = VGA with color display<br>0Bh = MCGA with monochrome display<br>0Ch = MCGA with color display |

**Table 4-10. VGA BIOS User Palette Table**

| Byte Number | Size | Contents |
|---|---|---|
| 0 | byte | Underlining flag: -1 = Off, 0 = Ignore, 1 = On |
| 1 | byte | Reserved |
| 2 | word | Reserved |
| 4 | word | Number of palette registers in the table |
| 6 | word | First palette registers in the table |
| 8 | dword | Pointer to palette register values |
| 0Ch | word | Number of DAC registers in the table |
| 0Eh | word | First DAC register in the table |
| 10h | dword | Pointer to DAC register values (table has 3 bytes per RGB register) |
| 14h | bytes | List of video modes terminated by 0FFh |

# Part II

## *SuperVGAs*

# Introduction

SuperVGAs (enhanced VGA-compatible products) are no different in architecture than the standard IBM VGA. Each VGA manufacturer has simply expanded the architecture to accommodate new display modes, and added proprietary features and options. Understanding SuperVGA really means understanding the proprietary extensions and features that have been added by each manufacturer.

Today, several dozen manufacturers build SuperVGAs. All of these products, however, are based on VLSI VGA chips (integrated circuits) from a small number of chip suppliers. The character of a particular VGA adapter is mainly determined by the VLSI device that the product is based on.

Fortunately, a great deal of similarity exists between SuperVGA implementations from different manufacturers. Programming is basically the same for all types of SuperVGAs. Some tailoring of software is required for the particular VLSI VGA device being used, especially for display modes that require the use of display memory paging. While most SuperVGAs include some form of memory paging mechanism, the actual memory paging scheme depends on the particular VGA chip being used. Some paging schemes are more powerful and flexible than others.

Chapter 5 is dedicated to a discussion of the common features and extensions of SuperVGA products. In chapters 6 through 9, programming examples are given to demonstrate basic programming concepts that are common to all SuperVGAs. A chapter is included for each type of high resolution VGA graphics mode: 2-bit pixels (four-color modes), 4-bit planar pixels (sixteen-color modes), and 8-bit packed pixels (256-color modes). These programming examples will reference manufacturer dependent constants and subroutines that can be used to tailor the routines to a particular board.

Chapters that follow the programming examples contain in-depth descriptions for each of the most popular VLSI VGA chips, one chapter per device, with a description of a typical SuperVGA board that uses that device. Sample code is provided to allow each device to be used with the programming examples of the earlier chapters. These descriptions were selected and organized to provide useful information that applies to the majority of VGA products that are currently available.

# 5

# Architecture of the SuperVGA

# Introduction

A lengthy discussion of SuperVGA architecture is actually not necessary; the basic architecture of the SuperVGA is the same as that of the standard IBM VGA and includes the same five major functional blocks: CRT Controller, Sequencer, Attribute Controller, Graphics Controller and Display Memory. All standard VGA functions, BIOS services, and registers, as described in chapters 1 through 4, are preserved. Detailed descriptions and programming examples for the standard features of VGA can be found in our earlier text *Advanced Programmer's Guide to EGA/VGA*.

The most significant additional feature of all SuperVGAs is the capability to display higher resolutions and more colors than the standard IBM VGA. Popular extended display modes are described in this chapter.

A key requirement for this extended display capability is the ability to address larger amounts of display memory than the standard VGA can accommodate, which is usually achieved with some type of memory paging mechanism. While paging schemes vary between manufacturers, the basic principle remains the same. Much of this chapter is dedicated to a description of memory paging.

SuperVGAs usually include other software support in the form of BIOS upgrades and application software drivers to support extended display modes. These and other added features will be discussed in this chapter, as well as the chapters which discuss specific products.

# Mapping of Display Memory

## Host Address Space / Host Window

VGA drawing operations are performed by the system processor, which reads data from and writes data to the display memory. To accomplish this, the display memory is mapped to a specific segment (or segments) of the host processor memory address space. This is sometimes referred to as the ***host window*** to display memory.

Table 5-1 shows the standard organization for the first megabyte of addressable system memory in IBM-compatible computer systems.

**Table 5-1. PC memory map**

| Address | Contents |
|---|---|
| F000:FFFF | |
| F000:0000 | BIOS ROM |
| E000:0000 | LAN, Tape Backup, EMS, ... |
| CC00:0000 | Other add-on card BIOSes |
| C800:0000 | XT Disk BIOS |
| C000:0000 | VGA/EGA BIOS ROM |
| B800:0000 | CGA/VGA/EGA Display Memory (Second page of Hercules RAM) |
| B000:0000 | MDA/VGA/EGA Display Memory (First page of Hercules RAM) |
| A000:0000 | VGA/EGA Display Memory |
| | Transient Program Area (User memory) |
| | Resident part of COMMAND.COM |
| | Disk buffers, Installable Drivers, ... |
| | DOS Kernel |
| 0000:0400 | BIOS Data Area |
| 0000:0000 | Interrupt Vectors |

The host window used by the VGA varies depending on the mode of operation. Table 5-2 contains the standard host windows and sample modes using each window.

**Table 5-2. VGA host windows**

| Host Address | Contents |
|---|---|
| C000:0000h - C000:5FFFh | IBM VGA BIOS ROM (C000:7FFF for most VGAs) |
| B800:0000h - B800:7FFFh | Color Text (Mode 3) |
| B000:0000h - B000:7FFFh | Monochrome Text (Mode 7) |
| A000:0000h - A000:FFFFh | VGA Graphics (Modes D, E, F, 10,...) |
| A000:0000h - B000:FFFFh | Extended Graphics (A000:FFFF for most modes) |

For text modes, which require relatively little data to be moved, a 32K space is used. In graphics modes, where much more data is required, a 64K space is used. When all four VGA color planes are used, this gives the processor access to 256K of display memory (4 x 64K).

As screen resolution and number of colors increase, the amount of display memory that must be accessed by the processor also increases. In some high resolution modes, more than 256K of display memory must be accessed.

A simple way to gain access to more display memory is to increase the size of the host memory space used by the VGA from 64K to 128K, using the memory address space from A000:0 to B000:FFFF. This standard VGA option, which is selected via the

Miscellaneous Register of the Graphics Controller, is both convenient and efficient but has the limitation that it interferes with other co-resident display adapters such as MDA, CGA, or Hercules. No IBM standard display modes use this 128K host window. An alternative way to access more than 64K is to use a display memory paging scheme.

## Memory Planes vs. Memory Pages

The standard IBM EGA and VGA include 256K of display memory. To allow processor access to the full display memory through a 64K host window, the display memory is divided into four memory planes (4 planes x 64K per plane = 256K). Memory planes are illustrated in Figure 5-1, and are described in detail in our earlier text, *Advanced Programmer's Guide to the EGA/VGA*.

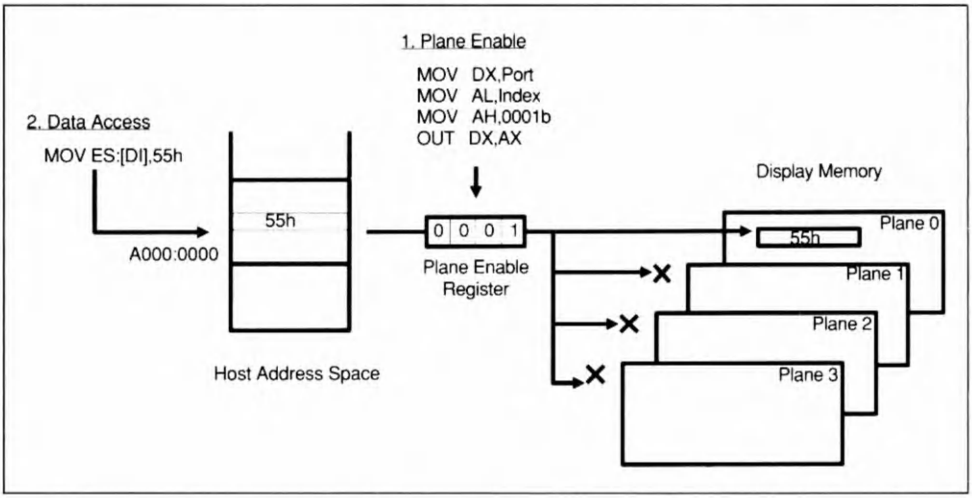


**Figure 5-1. Plane selection**

SuperVGAs may be configured with either 256K, 512K or 1024K of display memory. To allow processor access to this larger display memory through a 64K host window, most SuperVGAs include an added memory paging mechanism to allow a section of display memory to be mapped to the processor.

## Display Memory Paging

Display memory paging operates in a manner similar to the paging system used with expanded memory boards (also called EMS or LIM/EMS memory after Lotus Intel

Microsoft Expanded Memory Specifications). Before transferring data to or from display memory, an application program must first select the proper memory page by loading the page number into a page select register. This process is illustrated in Figure 5-2.

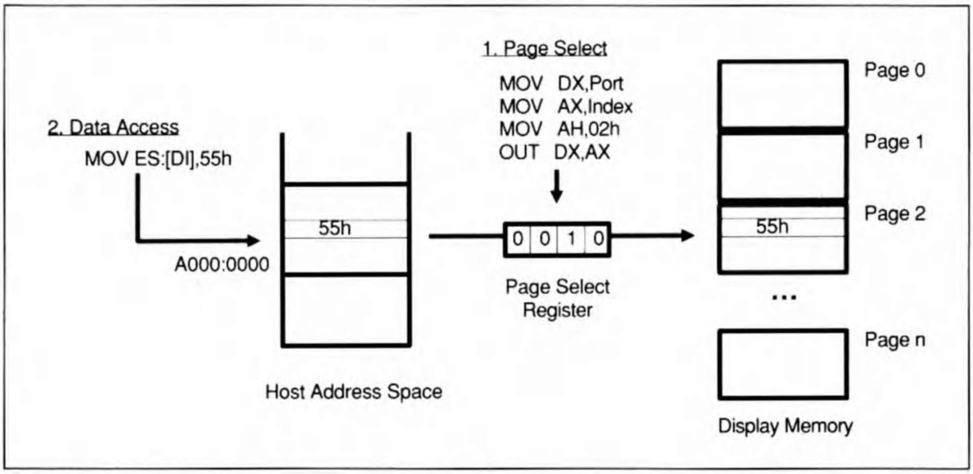


**Figure 5-2. Page selection**

Display memory paging mechanisms vary between SuperVGA manufacturers. In later chapters which concentrate on manufacturer specific features, programming examples titled Select_Page, Select_Read_Page and Select_Write_Page explain how to use the paging scheme for each of the popular SuperVGA devices.

Documentation from a particular manufacturer may refer either to memory paging or to memory banking; the concept is the same in either case. This text will use the term memory paging, following the precedent that was set for expanded memory boards.

The size of display memory pages varies between different VGA products, or even between modes of the same product. 32K and 64K are common page sizes. The granularity of the page starting address (the minimum increment with which the starting memory address for a page can be specified) also varies, and may be smaller than the actual page size. A large page size with small granularity is desirable. Figure 5-3 shows the effects of page size and granularity. A finer granularity requires more bits in the paging register, as shown in Table 5-3.

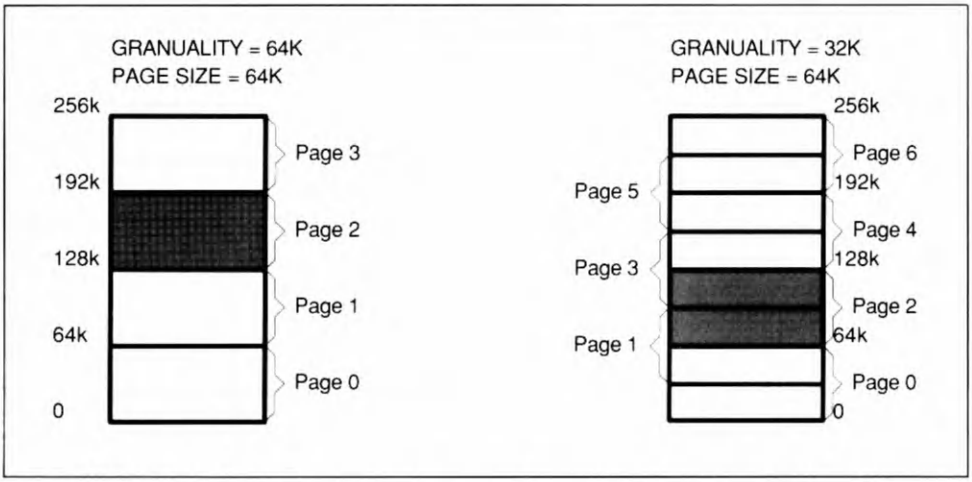


**Figure 5-3.** **Granularity and page size**

**Table 5-3.** **Granularity vs. bits needed for Page Select register**

| | **Bits Needed** | |
|---|---|---|
| **Granule Size** | **512K RAM** | **1024K RAM** |
| 1K | 9 bits | 10 bits |
| 4K | 7 bits | 8 bits |
| 32K | 4 bits | 5 bits |
| 64K | 3 bits | 4 bits |

Display memory paging schemes fall into three broad categories according to the number of simultaneous pages supported, and the types of access supported (Read, Write or Both) for each page.

## One Display Memory Page

In the simplest implementation of display memory paging, only one memory page may be selected at a given time. Functions that require data to be transferred from one area of display memory to another, such as BITBLT operations, can be difficult to perform using this scheme if data must be transferred between pages. Such transfers become a four step process: select source page, read data, select destination page, write data.

To minimize the amount of page switching required in such cases and to allow the use of block move (REP MOVSB) instructions, data can be transferred using a tempo-

rary buffer in host memory. A BITBLT operation using this approach is illustrated in Figure 5-4.

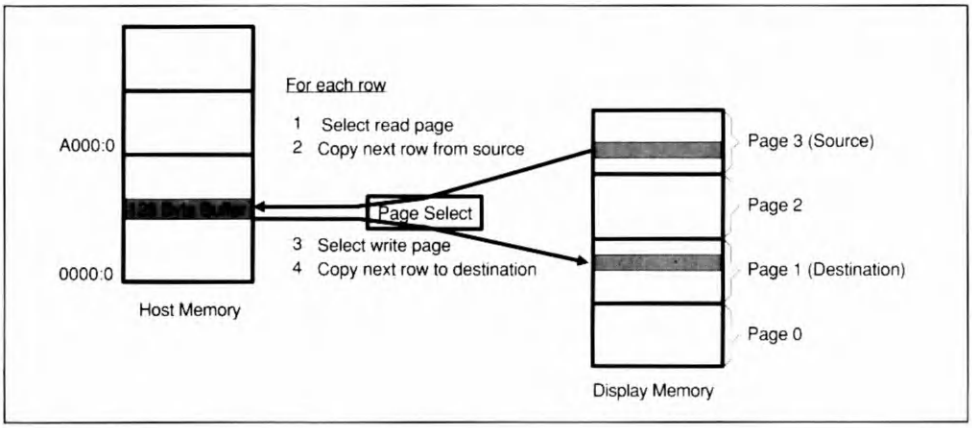


**Figure 5-4. BITBLT with one page**

## *Two Simultaneous Memory Pages, One Read-only and One Write-only*

A second approach to display memory paging allows two separate pages of display memory to be selected simultaneously, one page being read-only and the other page being write-only. Both pages are mapped at the same host memory address. This is illustrated in Figure 5-5.

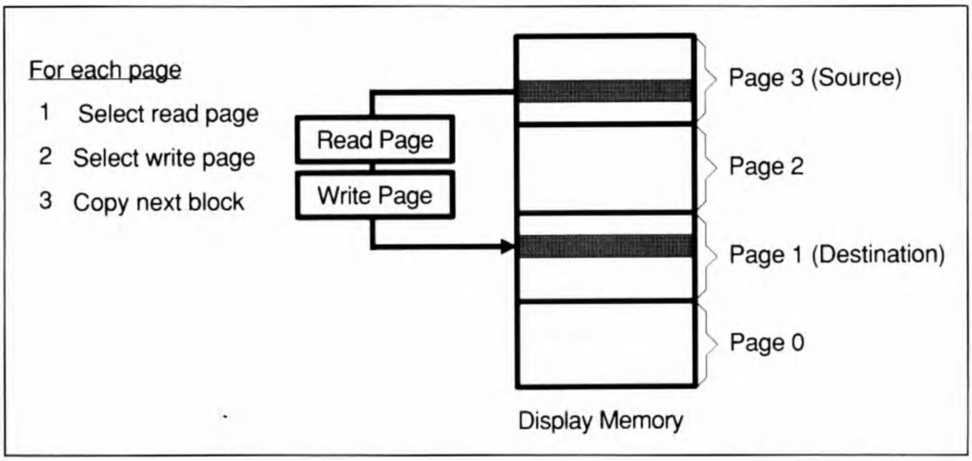


**Figure 5-5. BITBLT with Separate Read & Write Page**

This implementation permits fast data transfers from one page of display memory to another, since block move instructions (REP MOVSB) can be used to move data directly from one page to another. This approach also has limitations, however; it offers no advantage for BITBLT operations that perform logical operations (AND, OR, XOR, etc.) between source data and destination data, since the destination page is write only. An intermediate buffer in host memory is still required.

### Two Fully Independent Memory Pages

A more flexible approach to memory paging permits two fully independent memory pages to be selected simultaneously at different memory addresses. Using this scheme, BITBLT operations with logical functions can be performed with a minimum of page switching, and without an intermediate buffer in host memory. This approach also has disadvantages, however; since the two pages must reside at different host memory addresses, the host window must be twice as large, or the page size must be cut in half. Expanding the host window from 64K to 128K causes conflicts with secondary display adapters. Reducing the page size below 64K complicates algorithms that must detect page boundaries.

## Graphics Programming with Paged Display Memory

Accessing display memory through a memory paging mechanism causes an inevitable degradation in drawing speed. This degradation can range from negligible to severe, depending on how the drawing routines are written. As a drawing routine advances through display memory, it is important to: 1) minimize the frequency with which it must check for page boundaries, 2) minimize the number of instructions used in boundary checks, and 3) perform page selection only when required.

For some drawing algorithms, it is possible to compute where page boundaries will be crossed, then divide the drawing operation into two or more steps (one step for each page). Page boundary checks are then not needed during the repetitive inner loop of the drawing function.

## Page Boundary Detection

For efficiency, drawing algorithms that move through a range of x and y coordinates usually will not repetitively translate x,y coordinates into display memory addresses. This translation is performed only once to initialize the drawing routine; afterward, the drawing algorithm advances in the x or y direction simply by incrementing or decrementing the display memory address by a constant value. Such algorithms are referred to as DDAs (Digital Differential Analyzers).

As memory addresses are incremented or decremented, periodic checks must be made to detect the crossing of page boundaries. To maximize drawing performance, these checks must be designed carefully.

If the display memory page size is 64K, the JC (jump on carry) instruction of the processor can be used to efficiently check for page boundaries; for example, the following code can be used to detect a page boundary during y coordinate updates (Video_Pitch must be positive):

```
     ADD  SI,Video_Pitch          ;advance to next scan line
     JNC  Skip_Page_Select        ;skip page select if not needed
     CALL Select_Next_Page        ;select next page if needed
Skip_Page_Select:
     ...
```

For better efficiency, the jump instruction on every check can be avoided until a page boundary is detected (which occurs fairly infrequently):

```
     ADD  SI,Video_Pitch          ;advance to next scan line
     JC   Next_Page               ;check for page boundary
Return_Label:
     ...
```

or

```
     SUB  SI,Video_Pitch          ;decrement to previous scan line
     JC   Prev_Page               ;check for page boundary
```

When incrementing (or decrementing) the x coordinate, an increment (INC DI) or decrement (DEC DI) instruction will not update the carry flag. For page boundary detection using the carry flag, INC DI must be replaced by ADD DI,1 and DEC DI must be replaced by SUB DI,1. MOVS and STOS instructions also do not update the carry flag.

The time consuming jump is taken only when a page boundary is detected. The target of the jump (Next_Page or Prev_Page) must update the page select register, then jump back to continue the algorithm. A more complete example of this technique can be found in the **Line** programming example shown in Chapter 7.

High level languages do not easily allow the carry flag to be tested directly, but a test for overflow can still be performed if Offset (the display memory address offset) and Video_Pitch are stored as unsigned 16-bit data types. In C, for example, the following code can be used:

```
     if ((Offset = Offset + Video_Pitch) < Video_Pitch)
                     Select_Next_Page();
```

When the page size is less than 64K, page boundaries can be detected by testing specific address bits in the memory address:

```
     add  si,Video_Pitch
     test si,8000h          ;Check for 32K page boundary
     jnz  next_page         ;If bit set, page boundary was crossed
```

In many cases, it is not necessary to test for a page boundary following every memory address increment. During BITBLT operations, for example, it is usually sufficient to check for page boundaries at the end of each scan line. Each scan line can then be moved in the most efficient manner, utilizing instructions such as MOVS and STOS where appropriate. This approach may require that the memory page and offset be adjusted to assure that a page boundary can never be crossed in the middle of a scan line; in some modes this is automatic (in 1024x768 resolution, for example, 64K page boundaries will never occur in the middle of a scan line).

In the **Scanline** programming example, only one check is needed to see if the scan line crosses a page boundary. If the operation (x0 + dx) causes overflow, the scan fill is split into two steps; otherwise a single REP STOS instruction is used to draw the entire scanline. At most only two page selects are needed; one at the start of the fill, and possibly one more at the page boundary.

When using the block move instructions of the 80286 processor (REP MOVS or REP STOS), 16-bit transfers (MOVSW and STOSW) are more efficient than 8-bit transfers (MOVSB or STOSB). To handle the possible odd last byte created by using 16-bit transfers, the following code can be used:

```
MOV     CX,Count        ;Fetch how many bytes to do
SHR     CX,1            ;Convert byte count to word count
REP     MOVSW           ;Move all words
ADC     CX,0            ;Set counter to do the possible odd byte
REP     MOVSB           ;Move the possible odd byte
```

When both DI and SI are initially even (word aligned), the transfer is even faster, since only half as many transfer cycles are required with MOVSW as with MOVSB.

It should be noted that 16-bit transfers may not be usable in 16-color planar modes if the VGA processor data latches are being used for Latched Writes, bit masking, or logical operations, since these latches are only 8 bits wide.

# Enhanced Modes

Enhanced display modes with higher resolution and more colors are the most important feature of the SuperVGAs. High resolution text modes that permit 132-column spreadsheets to be displayed are common, as well as high resolution graphics modes with 256 simultaneous color capability. Not all VGA boards support the same enhanced display resolutions, but certain resolutions have become de facto standards. These have mainly been determined by the capabilities of the displays that are available. Popular resolutions include 640x480, 800x600, and 1024x768 pixels. Since these

modes were developed as extensions to the basic VGA, there is usually a high degree of similarity in the way that they are implemented.

## Enhanced Text Modes

By varying both the resolution of the display and the size of a character cell, many different text modes can be supported.

Modes that display a wider screen (more characters per line) are useful for applications such as spreadsheets where many columns of data must be displayed. 132-column text is popular since it represents a standard width for computer printouts, but even at the highest resolutions characters are small and difficult to read in this format. 100-column and 120-column formats are also popular.

132-column text modes usually require more than 1000 pixels of horizontal resolution on the display. While this exceeds the published specifications for virtually all VGA class displays, characters are still readable on most of the popular VGA displays, although with some loss of quality.

Organization of display memory for enhanced text is the same as that for standard VGA modes (see chapter 2). Enhanced features include higher resolutions and additional attribute bits for font selection (see Figure 5-6).

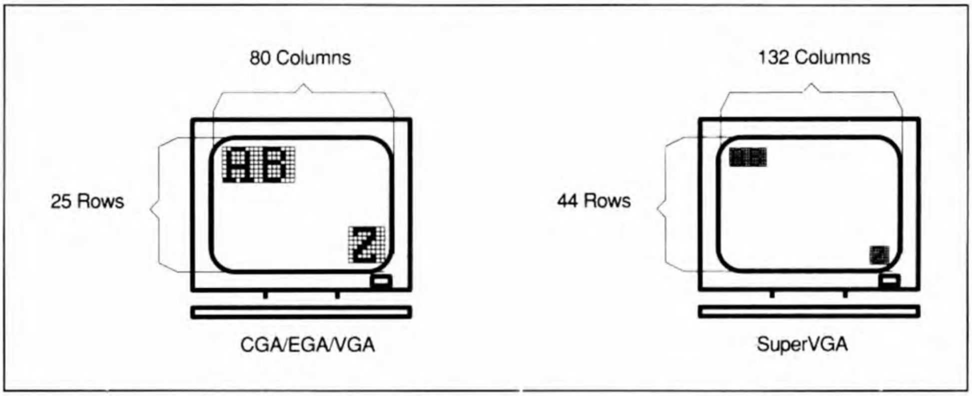


Figure 5-6. Enhanced text modes

## Enhanced Graphics Modes

Modes that offer 256 simultaneous colors at resolutions greater than the 320x200 pixels offered by IBM's mode 13h can be used to present full color photographic images with impressive fidelity. Modes that offer 16 colors at higher resolutions than the IBM VGA are popular for applications that involve fine visual details, such as CAD/

CAM and desktop publishing. Modes that offer high resolution with only 2 or 4 colors are popular for WYSIWYG (What You See Is What You Get) displays in desktop publishing, where resolution is important but colors usually are not. Figure 5-7 illustrates the enhanced graphics modes resolutions.

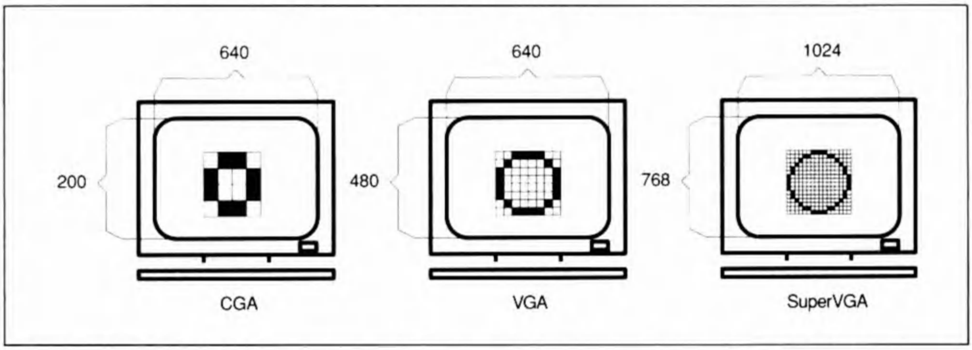


**Figure 5-7. Enhanced graphics modes**

## *640 x 400 256-Color Graphics*

This is a logical resolution for many adapters to support because it requires 256,000 Bytes of display memory, which is the amount of display memory included on every VGA product (262,144 Bytes). The resolution is also an exact multiple of the standard VGA mode 13h (320x200 256 colors.). This is the only common extended graphics mode which does not have square pixels (4:3 aspect ratio is needed to achieve square pixels on industry standard displays).

The implementation of this mode will usually resemble VGA mode 13h except that both the number of pixels per scan line and the number of scan lines are doubled. Since color planes are not used in 256-color modes (these modes use packed pixels), some form of memory paging is needed to make the full 256K of memory available to the processor. Display memory organization for this mode is explained in detail in chapter 8.

## *640 x 480 256-Color Graphics*

A resolution equal to the highest standard VGA resolution, with 256-color capability, makes this a logical mode for SuperVGAs. This mode can only be supported by VGAs that include at least 512K of display memory.

The implementation of this mode will usually resemble mode 13h except that the number of pixels per scan line is doubled and the number of scan lines is increased. Some form of memory paging is required to make the larger display memory available

to the processor. Display memory organization for this mode is explained in detail in chapter 8.

## 800 x 600 256-Color Graphics

This is the highest resolution that is available on most low cost (under $700) multi-frequency displays. It is also the highest possible 256 color resolution available on adapters with 512K of display memory. Full color photographic images can be displayed with remarkable fidelity at this resolution. This mode requires 480K of display memory, and usually resembles mode 13h except that the number of pixels per scan line and number of scan lines are both increased. Display memory paging is required. The organization of display memory for this mode is described in chapter 8.

## 1024 x 768 256-Color Graphics

This is the highest resolution found on SuperVGA cards today. Although some chip manufacturers claim the capability of resolutions up to 1280x1024, or 16-bit pixels (65,536 colors), as of this writing there are no SuperVGA adapters available with capabilities beyond 1024x768 with 256 colors. This mode requires 768K of display memory, and resembles mode 13h except that the number of pixels per scan line and number of scan lines are both increased. Display memory paging is required. The organization of display memory for this mode is described in chapter 8.

## 800 x 600 16-Color Graphics

This mode requires 240K of display memory, and is the highest 16-color resolution that can be supported using only 256K of display memory. It is also the highest 16-color resolution that can be supported without utilizing some form of memory paging scheme to allow the processor to access the full display memory. This resolution is also the upper limit of resolution on the original multifrequency displays.

The implementation of this mode usually resembles mode 12 (640x480 16-color graphics), except that both the number of pixels per scan line and the number of scan lines is increased. Display memory organization for this mode is described in Chapter 7.

## 1024 x 768 16-Color Graphics

This is the highest resolution that is commonly found in VGA products. Only the best VGA displays can support this resolution. Its implementation usually resembles mode 12 (640x480 16-color graphics), except that both the number of pixels per scan line and the number of scan lines is increased.

In order for a VGA adapter to support this resolution, its design must include two key elements: it must include at least 512K of display memory to accommodate the

larger screen, and it must have the capability to operate at a video rate of around 65 MHz (or 45 MHz for interlaced displays).

At this resolution, the screen contains 786,432 pixels which exceeds the number of pixels (524,288) that can be accessed in one 64K segment of memory. To make this large display memory accessible to the processor, some form of memory paging must be employed. The exact implementation varies depending on the manufacturer.

Some displays use interlacing to reduce the bandwidth requirement at this resolution. Some VGA boards support both interlaced and non-interlaced displays, some support interlaced displays only, and some support non-interlaced displays only.

Display memory organization for this mode is described in detail in Chapter 7.

### 1024 x 768 4-Color Graphics

4-color graphics are popular for desktop publishing, where color is usually not as important as the resolution. Limiting the number of colors allows higher resolutions to be supported without increasing the size of the display memory. This resolution is the highest resolution possible on boards with 256K of display memory. Fewer colors for each pixel can also result in improved performance since the processor has fewer bits to write during drawing operations.

Typical display memory organization for this mode is explained in Chapter 9.

## The BIOS

Ideally, all standard BIOS functions should be available in all of the extended modes of a SuperVGA. This is unfortunately not the case for most VGA boards. Virtually all SuperVGAs support the BIOS Mode Select function in all extended display modes; other BIOS support differs depending on the manufacturer.

Some manufacturers have extended the BIOS text functions to work in their extended text modes, but many VGAs do not support the ability to use BIOS text functions while in an extended graphics mode. These functions are especially difficult to support in high resolution graphics modes which require display memory paging.

As VGA complexity has increased, the size of the BIOS ROM has also increased. The EGA BIOS uses 16K of system memory in the address range from C000:0000 to C000:3FFF. The IBM VGA uses 24K of system memory in the address range from C000:0000 to C000:5FFF. Some manufacturers have taken even larger spaces for their BIOS; others have developed methods of expanding the BIOS without increasing its allocated ROM space.

Some SuperVGA boards use a paged BIOS ROM. Since ROM is not affected by memory write operations, a write operation to the BIOS ROM space is used to select the desired ROM memory page.

Other VGA boards locate some of their BIOS code in system memory. This method will also speed up the execution of BIOS functions, since system memory is normally faster than BIOS ROM memory. This can provide a measurable performance improvement during text operations.

The Video Electronics Standards Association (VESA) has defined a new set of BIOS functions which can be used to improve compatibility between different VGA products. To learn more about VESA, see Chapter 20.

# Other Features

Some VGA products offer other useful features such as hardware zoom (the ability to enlarge a section of the screen), or hardware support for a graphics cursor. This kind of added support can improve the overall performance of the board by reducing the overhead imposed on the system processor, providing the software is written to take advantage of these features.

## Application Software Drivers

New display modes are of little use if the software you are using doesn't support them. A lack of standardization between VGA vendors has hindered application software vendors who are adding support for new VGA modes to their products. VGA vendors have been forced to take on the task of supplying drivers for popular application programs. Drivers that are commonly supplied include Microsoft Windows, GEM, Ventura Publisher, Autocad, Lotus 1-2-3, Versacad, Word Perfect and Word.

## 16-bit Data Buses

The IBM EGA and VGA boards use an 8-bit data bus to communicate with the processor. It is a common practice when designing enhanced versions of IBM AT-compatible products to add the capability to operate on a full 16-bit data bus, on the theory that data transfers will be faster with a wider bus. In most popular applications, the increase in data transfer rate to display memory usually has little or no measurable effect on the performance of the system or display, and in some systems it may even conflict with other add-on boards. The 16-bit data bus has proven to be an effective marketing tool, however, and seems to help sell boards regardless of its merits. "sixteen bit" seems to have the same effect on buyers as the words "fuel injection" on a sports car.

For many types of drawing functions, most VGA graphics programmers will choose to restrict memory transfers to only 8 bits at a time to maintain compatibility with as many VGA boards as possible.

Some performance gain can be achieved by widening the VGA BIOS ROM from 8 bits to 16 bits. This improvement is only seen during BIOS text functions, however.

Text functions are normally not a performance problem, even in the slowest of systems.

## Automatic Display Detection

The IBM VGA, as well as many SuperVGAs, can automatically detect through the display interface cable whether the display that is attached is color or monochrome. Detection is done automatically by the VGA BIOS. If a monochrome display is detected, the video DAC registers are adjusted to convert color information into monochrome gray scale information.

Some VGA products do not include automatic display detection but use EGA style configuration switches instead. Newer VGA products will normally support display detection.

Automatic display detection makes system configuration easy, but it can have side effects. If a system is started with no display connected, the VGA may default to monochrome mode. This means that if you power up your system with the display unattached (or not powered on) then attach (or power on) the display, the display will be in a monochrome mode until the VGA is reset. Often this means recycling power on the system to reset the VGA to its proper mode.

# Adapter Identification

When writing software that must cope with different types of video adapters, it is often useful to be able to automatically detect what type of video adapter is being used. Unfortunately, there is no single test that can provide this information for all adapter types. A sequence of tests can often identify the adapter type:

1.) Video BIOS function 12h, subfunction 10h (return info on EGA/VGA configuration) can be used to identify if an EGA or VGA is present in the system. The BH register of the processor will be modified by this call if and only if an EGA or VGA BIOS is present:

```
MOV  AH,12h          ;BIOS function 12h
MOV  BL,10h          ;Subfunction 10h
MOV  BH,55h          ;Initialize BH for test
INT  10h             ;Make BIOS call
CMP  BH,55h          ;If BH is unchanged,
JE   No_EGAVGA       ;There is no EGA or VGA
```

2.) Video BIOS function 1Ah, subfunction 0 (Read Display Configuration Code) can be used to distingush between EGA and VGA presence. If a VGA is present, register AL will return a value of 1Ah:

```
MOV  AH,1Ah          ;BIOS function 1ah
MOV  AL,0            ;Subfunction 0
INT  10h             ;Do BIOS call
```

```
CMP  AL,1Ah         ;If al = 1ah,
JE   VGA_Found      ;A VGA is present
```

3.) The manufacturer of a particular VGA board can frequently be determined by examining the BIOS ROM area at memory address C000:0000 for copyright messages or signature bytes, or by testing for the presence of special "Extended" I/O registers, or by a combination of both. As an alternative, one can loop through known paging methods, and for each method attempt to fill several pages with its page number, until these page numbers can be reliably read back.

4.) After determining the manufacturer, it may be necessary to determine the version of the board or version number of the VGA chip to determine what features it will support. This type of test, however, becomes very device-specific and may even require a written agreement to be executed with the manufacturer to receive the required information.

Specific adapter identification methods are provided later in the text for those manufacturers that made such information available.

# Selecting a SuperVGA

When selecting a VGA adapter for a particular application, consider the following factors:

## Know Your Application

- Are particular high resolution modes most important?
- Are 256-color modes important?
- Are there particular application programs that the VGA must be compatible with?
- Will you be writing or adapting software yourself?
- Will the vendor provide you with programming info (clones often do not)?
- What other adapter boards will be resident in the system?
- Are there any potential address conflicts in the modes you need?
- Is support needed for TTL displays?

## Know Your Operating System

Not all SuperVGAs will run with operating systems other than DOS. If you plan to use OS/2, Unix, Xenix or other operating systems, make sure you test the board first with the particular operating system. Little or no support is provided to permit the use of extended display modes with Unix or any of its derivatives.

## Evaluate Compatibility

- Are EGA, CGA, or MDA emulations supported and are they needed?
- Do BIOS functions work in the extended modes?
- Are extended display modes closely patterned after IBM standard modes?
- Will there be address conflicts with other boards in your system?
- Do memory and I/O spaces conform to IBM standards?
- Are drivers provided for your application taking full advantage of the board?

Some VGA boards use a full 128K memory address space in high resolution graphics modes. This can create incompatibilities with other video adapters.

## Know Which Displays are Supported

Many VGA boards are limited in the choice of monitors they support. For example, some "Hercules compatible" VGAs will not run with monochrome TTL displays. Some VGAs support 1024x768 interlaced or non-interlaced, but not both.

## Evaluate Features

- Are any useful application software drivers provided?
- Can you utilize any vendor specific features (such as hardware zoom or hardware graphics cursors)?

## Evaluate Performance

- Do the application software drivers perform well?
- Does the display memory paging scheme perform well?
- Does it have a 16-bit bus?
- Does it run the video BIOS in RAM?
- Will it run in high speed systems (20MHz and above)?

IBM Compatibility

Manufacturers of IBM-compatible equipment have had a difficult job trying to maintain compatibility with IBM in the last few years. As IBM implements more and more of their display circuitry in proprietary VLSI integrated circuits, the task of designing a clone has become formidable. IBM even has two versions of their VGA chip, one for use on system boards and one for use as an add-in board.

The IBM EGA was introduced in 1985, and it was a full year later that companies such as Chips and Technologies and Tseng Labs were able to produce the chip sets (integrated circuits) required for compatible products to be produced. Shortly

thereafter, when IBM unveiled the VGA, some companies began making premature claims of VGA compatibility. They placed labels such as "VGA BIOS compatible" on products that were not much more than EGA boards with upgraded BIOS ROMS. After several more months of intense engineering, true VGA-compatible products became available.

Even in its enhanced modes of operation, a SuperVGA board should remain true to the IBM standard. Ask questions such as these to evaluate IBM-compatibility: Is it IBM-compatible even in non-IBM modes? Are the BIOS functions supported in the new modes? Is the full register set useable? Does the memory map resemble that of any of the standard modes?

# 6

# *Programming Examples Overview*

# How the Programming Examples are Organized

Since the focus of this book is on the enhancements that separate SuperVGAs (enhanced VGA boards) from the standard IBM VGA, the programming examples will concentrate on how to utilize the enhanced features and extended display modes of the SuperVGAs. To learn more about standard VGA features see our previous text, *Advanced Programmer's Guide to EGA/VGA*.

Despite the lack of standardization among VGA suppliers in the way that enhancements have been added, most extended SuperVGA display modes are closely patterned in structure after standard IBM display modes. This fortunate circumstance eases the burden of programming for SuperVGAs, as well as the task of documenting it (for which the authors are grateful). In addition, display modes with a particular color capability but different resolutions tend to be very similar in structure.

Because of these basic similarities between the extended display modes of SuperVGA products from various vendors, the same basic drawing algorithms apply to most SuperVGA products with only minimal modification. By separating board-specific functions from the basic drawing algorithms (via procedures and global variables), we have centralized the bulk of our programming examples into common sections that apply to most extended display modes with minimal modifications.

Several versions of each drawing routine are provided according to the type of memory organization used for each display mode. SuperVGA memory organization is divided into three basic types: 256-color graphics, 16-color graphics, and 4-color graphics. In chapter 7, titled "Programming Examples—256-Color Graphics," programming examples are provided that can be applied to all 256-Color graphics modes. Chapter 8, titled "Programming Examples—16-Color Graphics," gives programming examples for 16-color graphics modes using planar pixels (ATI is the only manufacturer that offers 16-color mode using packed pixels) Chapter 9, "Programming Examples—4-Color Graphics," shows programming examples for 4-color modes (three different approaches are presented there).

For each basic memory organization, examples are given showing how to draw basic graphics primitives such as pixels, lines and rectangles, and how to perform BITBLT transfers. Also included are routines to draw and erase a software graphics cursor, load the color palette, and move a screen image to or from a file on disk. Because of the length and complexity of some of these functions, some programming examples are explained but not completely listed in the text. Complete versions of all examples can be found on the diskette that accompanies this book.

All drawing algorithms are written in assembly language. The test program, used to exercise the drawing routines, is written in the C language. Assembly language routines are written assuming that input parameters will be placed on the stack before the routine is called, conforming to the convention for C-callable subroutines. For infor-

mation regarding recent updates to the diskette and support for other languages see the READ.ME file on the diskette.

All drawing routines assume that the display adapter is already initialized to the appropriate display mode before the drawing routine is invoked.

# What is on the Diskette

The diskette of programming examples is structured to follow the programming examples in the text. Examples have been divided into two groups, the board-independent examples and the board-dependent examples, as indicated in Figure 6-1.

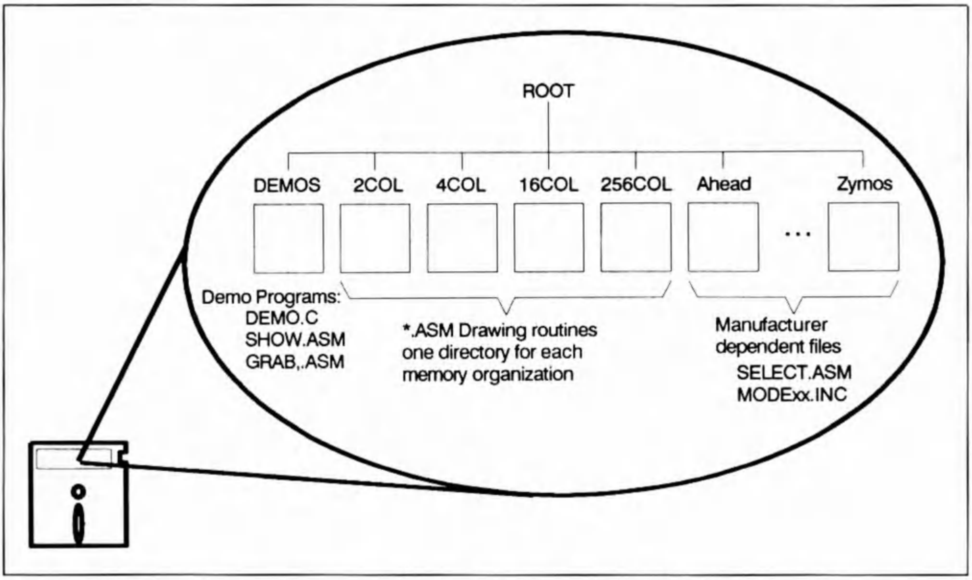


**Figure 6-1. Diskette organization**

Files in the directory DEMOS are demonstration and test programs DEMO.C, SHOW.ASM, and GRAB.ASM, and sample scanned images PICTUREx.IMG. Files in directories 256COL, 16COL, 4COL, and 2COL contain drawing routines that are independent of any particular board (one directory for each memory organization type). Other directories contain files with board-dependent and mode-dependent procedures.

Each directory contains a file named SELECT.ASM containing mode and page selection procedures, files named MODExx.INC containing mode-dependent constants, and make files named DEMO, DEMO.LNK, SHOW and GRAB.

For more up-to-date information on the content of the diskette, please read the file READ.ME found in the root directory of the diskette.

# How to Use the Programming Examples

Programming examples consist of 1.) a set of common drawing routines, and 2.) a set of board-specific special examples. All drawing routines are designed to be configurable to run at various resolutions and on various different VGA products. The drawing routines reference a number of board- and mode-dependent variables, and call board- and mode-dependent subroutines.

Board-specific examples, variables, and routines are provided in later chapters that are dedicated to superVGA products from specific manufacturers. The general drawing routines are described in this chapter.

Included on the program diskette is the demonstration module DEMO.C which is provided to illustrate the use of each of the drawing routines. To demonstrate the drawing routines on a wide range of boards and with all major graphics modes, the demonstration modules have been set up to be linked as indicated in Figure 6-2.

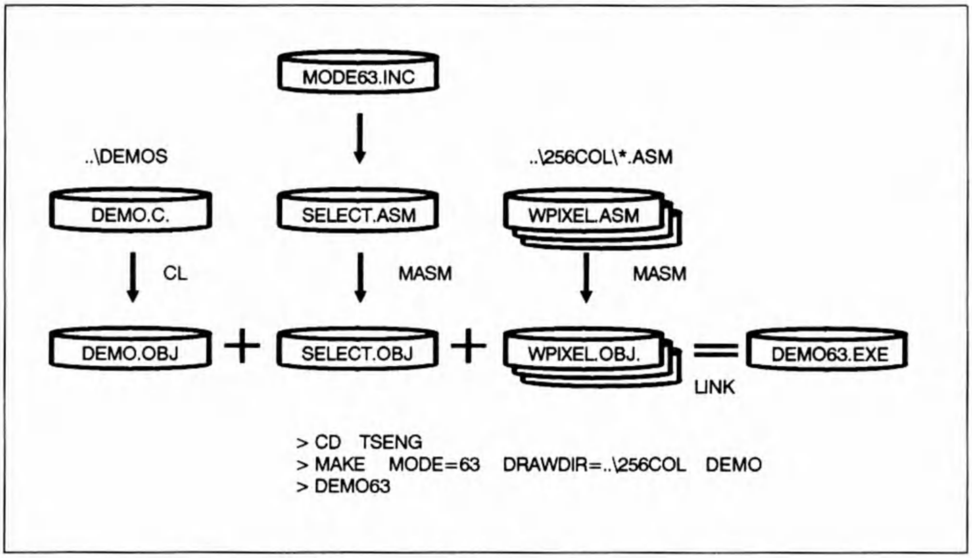


**Figure 6-2. Building the DEMO.EXE program**

Each version of the DEMO.EXE program can be built using the MAKE.EXE utility provided with Microsoft and Turbo language products. For each board manufacturer represented in the text, a Make file DEMO (no extension) is provided which is used with

the MAKE.EXE utility to build the program. Use the CD (change directory) command to change to the directory for the desired board manufacturer and then invoke the MAKE.EXE utility.

Two command line parameters are need to properly use the Make file. The MODE macro determines the mode to be used. The DRAWDIR macro determines which set of drawing routines to use (4, 16 or 256 color). For example, to build a program DEMO62.EXE for the 640x480 256-color mode on ATI boards use the following commands:

```
CD ATI
MAKE MODE=62 DRAWDIR=..256COL DEMO
```

These commands will cause the file MODE62.INC to be included in the file SELECT.ASM (described in the next section), and link it with DEMO.C and with drawing routines for 256-color memory organization. Batch files are provided on the diskette to invoke the MAKE.EXE process for each mode.

## Board- and Mode-Dependent Variables

Board- and mode-dependent variables are initialized during the build process from an appropriate include file MODEXX.INC. The name of the proper include file is provided as a parameter to the MAKE.EXE utility. For example, to build the demo program for the ATI board the following commands can be used:

```
CD ATI
MAKE MODE=62 DRAWDIR=..256COL DEMO
```

The constants in the Include file are used to initialize variables referenced by the drawing routines. Table 6-1 contains a list of constants in the Include file, and the corresponding variables initialized with the constants in the SELECT.ASM file.

**Table 6-1. Mode-dependent constants and corresponding variables**

| Constant | Variable used by Drawing Routines |
|---|---|
| SCREEN_PITCH | Video_Pitch |
| SCREEN_WIDTH | Video_Width |
| SCREEN_HEIGHT | Video_Height |
| SCREEN_PAGES | Video_Pages |
| CAN_DO_RW | Two_Pages |
| GRAPHICS_MODE | (Used in INT 10 video service) |

Meanings of these constants are as follows:

**SCREEN_PITCH** equals the logical length (in bytes) of a scan line. Adding the value VIDEO_PITCH to a memory address is equivalent to advancing vertically by one scan

line on the screen. For 800x600 planar mode, this value will normally be 100 decimal (800/8); for 1024x768 planar mode, a value of 128 decimal (1024/8) is normally used.

**SCREEN_WIDTH** equals the number of pixels in one scan line. In 256-color modes this is same as SCREEN_PITCH.

**SCREEN_HEIGHT** equals the number of scan lines visible on the screen. For example in 640x480 graphics modes, this will be set to 480.

**SCREEN_PAGES** equals the number of memory pages needed to clear the entire screen. The clear screen routine will clear 64K per page for pages 0 through Video_Pages - 1.

**CAN_DO_RW** is a flag that indicates, when true (nonzero), that the display memory paging mechanism for the board can support two simultaneous memory pages (one for reading and one for writing). Bit 0 is used to indicate that separate read and write pages are available at the same 64K host window. Bit 1 is used to indicate that two independent pages are available, each at its own 32K host window.

**GRAPHICS_MODE** is the mode number used to invoke a graphics mode. This number is normally the number used in the Mode Select service of the INT 10h BIOS call.

For each board there are several MODExx.INC files, one for each memory type. Files for other modes can be constructed from these, using the existing files as templates.

## Board- and Mode-Dependent Routines

Board-dependent routines referenced by the drawing routines are contained in the module SELECT.ASM which is in the appropriate board-dependent section. The SELECT.ASM module contains the following routines:

**Select_Graphics** selects a graphics mode specified in the include file MODEXX.INC (mode XX is specified as a command line parameter in the MAKE directive). Typically this will be a call to Mode Select service of INT 10h BIOS calls.

**Select_Page** selects a specified page of display memory. Page selection is usually performed by the simple procedure of outputting the desired page number to an I/O port, but the way in which the page selection port is addressed varies from vendor to vendor.

**Select_Read_Page** selects a specified 64K page of display memory for reading, on boards that support separate read and write memory pages, and selects 32K page A for boards that support two independent pages.

**Select_Write_Page** selects a specified 64K page of display memory for writing on boards that support separate read and write memory pages, and selects 32K page B for boards that support two independent pages.

The module SELECT.ASM also contains the global variables **Video_Pitch, Video_Width, Video_Height** and **Two_Pages** initialized by constants from include file MODExx.INC, described in the previous section and shown in Table 6-1.

For programming convenience, two other global variables are defined in the module SELECT.ASM:

**Graph_Seg** is the memory segment address where display memory is mapped to the host. This is normally set to segment A000h.

**Line_Buffer** is the address of a buffer in host memory which is large enough to buffer one scan line of pixel data. This buffer is used by BITBLT procedures as an intermediate buffer for boards which do not have dual page capability.

Listings for these procedures can be found in later chapters of the book that provide board-dependent information for individual manufacturers.

## Computing which Page to Select

Drawing examples in this text take input parameters in terms of x,y screen coordinates, and translate these coordinates to addresses in display memory. Coordinates **X,Y** are translated to **Page:Segment:Offset**, and for some modes a **Mask** value, where:

**X and Y** are the coordinates of a pixel on the display, with pixel 0,0 at the upper-left corner of the screen, x increasing to the right, and y increasing down.
**Page** is a page number in display memory. This value depends on the page size and granularity of a given mode used for the board.
**Segment** is the host memory address segment (usually fixed at A000).
**Offset** is the host memory address offset within the segment.
**Mask** determines bit position(s) within a byte for a particular pixel.

The computation required to perform this translation will usually be of the form:

Page:Offset = (Y * Video_Pitch + X / Pixels_Per_Byte) / Page_Granularity

where:

**Page** is the quotient of the expression.
**Offset** is the remainder of the expression.
**Video_Pitch** is the logical length of a scan line (in bytes).
**Pixels_Per_Byte** is 8 for planar pixel modes , 1 for 8-bit packed pixel modes.
**Page_Granularity** is the increment, in bytes, between successive memory pages.

For example, in 256-color drawing routines, with 64K pages, page and offset can computed using code similar to the following:

```
;Convert x,y to Page:Offset
MOV         AX,Y_Coord          ;Fetch y coordinate
MUL         CS:Video_Pitch      ;Compute offset of first byte in y-th line
ADD         AX,X_Coord          ;Compute offset of x-th byte within y-th line
ADC         DX,0                ;Add overflow from previous addition
;Save computed values
```

```
MOV             SI,AX               ;Save offset of the pixel
MOV             PageNumber,DL       ;Save page number
;Select page
MOV             AL,PageNumber       ;Select page
CALL Select_Page
;Fetch data
MOV             DS,CS:Graf_Seg      ;Fetch segment of display memory (A000)
LODSB                               ;Fetch pixel value at x,y
```

It is important to be aware that the page granularity may not be the same as the page size. Ideally, page granularity should be as small as possible, allowing pages to be selected with the greatest flexibility, while the page size should be as large as possible so that page switching can be minimized. Page size and granularity are similar concepts to the segment size and granularity of Intel 80x86 processors, where segment size is 64K, and segment granularity is 16 bytes.

In order to design a common set of drawing routines that can be usable for all SuperVGAs, we have elected to use a page size and granularity of 64K for all programming examples. This may not be the optimum case for any particular board or display mode, and more efficient versions of the routines can be written for some SuperVGAs, but this generalization greatly simplifies the drawing routines, and allows for a smaller set of examples.

# Drawing Routines

In this section is a brief description of each of the drawing routines available on the diskette. Each routine has several versions, one for each memory organization. Listings for most of these routines are included for each memory organization type in chapters 7 through 9. All routines are include on the diskette.

Accessing display memory through a memory paging mechanism causes an inevitable degradation in drawing speed. This degradation can range from negligible to severe, depending on how the drawing routines are written. As a drawing routine advances through display memory, it is important to: 1.) minimize the frequency with which it must check for page boundaries, and 2.) perform page switching efficiently (see the section "Graphics Programming with Paged Display Memory," in Chapter 5). Each of the drawing routines here has been selected to demonstrate a particular technique. The following examples provide representative techniques needed for many common drawing functions.

## Write Pixel

Writing a single pixel is the most basic drawing function, and makes a useful example. For practical drawing algorithms, however, pixels are usually not written one at a time. The Write Pixel function is too slow to use in a practical drawing algorithm.

Procedures Write_Pixel and Read_Pixel illustrate the conversion of (x,y) pixel coordinates to (Segment:Offset:Page) display memory addresses, and show how to

access individual pixels. In 256-color modes this process is reduced to a computation of Page and Offset; in other modes bit masks must be computed and plane enable registers must be manipulated.

## Read Pixel

Read Pixel, like Write Pixel, is a useful programming example that shows how to access individual pixels but is too slow for use in practical drawing algorithms.

## Draw Solid Line

While the task of drawing a straight line between two points may seem simple, fast and efficient, line drawing algorithms are actually quite complex. The line drawing routine shown here is based on Bresenham's Algorithm, which is described in detail in the text *Fundamentals of Interactive Computer Graphics* by Foley and Van Dam.

This procedure provides a good example of how page boundaries are detected in incremental algorithms. For 64K pages, page boundary crossings occur when the address offset 'overflows' or 'underflows'; it is sufficient to check the carry flag after the address offset is updated. Care must be taken to use the proper instruction to update the offset; the instructions INC and DEC will not set the carry flag but ADD and SUB will. When a negative offset is added or subtracted, the role of the carry flag is reversed.

## Draw Scan Line

This routine draws a horizontal line between two specified points. It is much faster and more efficient than the generalized line drawing routine shown above, and is useful for performing operations such as polygon fills. It also shows how to avoid checking for page boundaries after every pixel is drawn. This routine takes advantage of the fact that within any scan line there will be at most one page boundary to be crossed.

## Fill Solid Rectangle

Rectangle filling is the simplest form of fill operation. A rectangle of a specified size is filled with a given color. In planar modes this routine illustrates how to efficiently handle partial bytes at the leading and trailing edges if the rectangle.

## Copy Block

Bit block transfer operations (BITBLTs) move a rectangle of pixels from one part of display memory to another. Graphical user interfaces, such as Microsoft Windows and GEM, rely heavily on the use of BITBLT operations. BITBLT routines can be very com-

plex because of the number of cases that must be considered. The block move may move data from off-screen memory to on-screen, from on-screen memory to off-screen, or from one on-screen location to another on-screen location. If source or destination are off-screen, then source and destination memory may have a different pitch (logical line length). If both are on-screen, then source and destination may overlap. In this case, care must be taken so that source data is not destroyed while the move is in progress. In addition, the BITBLT may involve a logical operation between source data and the background data in the destination rectangle.

This example works for only the simplest case of BITBLT, where both source and destination are in display memory. In 256-color modes, no logical operations are performed. In planar modes, only the built-in operations COPY, XOR, OR, and AND are supported. It is still necessary to check for overlap between source and destination, and to check for page boundary crossings. The BITBLT procedure provides an example of how to copy a block of data, where the source may lie in a different page than the destination, and how to perform the copy with a minimal number of page select operations.

## Set Cursor, Move Cursor, Remove Cursor

Graphics cursors, such as arrows, crosshairs, or other shapes, are commonly used in graphical environments. Because cursors are frequently moved, it is important that they are drawn and erased efficiently. Three basic routines are needed to maintain a graphics cursor:

**Set_Cursor**: Stores a user-defined cursor shape, and sets a flag indicating that the cursor is active.

**Move_Cursor**: Restores the background data at the previous cursor location, saves the background data at the new cursor location to off-screen memory, and draws the cursor at the new location.

**Remove_Cursor**: Restores the background data at the previous cursor location; resets flag to indicate that cursor is no longer active.

When available, this module is replaced, in the make file DEMO, by the board-dependent module HWCURSOR.ASM, to demonstrate use of a hardware cursor on boards that support it.

## Load DACs

For 256-color modes, selection of screen colors can be modified by altering the color lookup table in the video DACs. This can be accomplished through a BIOS function call, or by directly loading the registers as shown in this example.

## Load Palette

For 16-color and 4-color modes, selection of screen colors is normally done by altering the palette registers in the Attribute Controller. This can be accomplished through a BIOS function call, or by directly loading the registers as shown in this example.

## Write Raster, Read Raster

For 256-color modes we have included the routines Read_Raster and Write_Raster. These two routines are used by the GRAB.COM program to save an image in display memory into a file, and by the program SHOW.EXE to display an image from a file. Several scanned images are provided on the programming diskette. Each image file contains a 768-byte color table which is used to load the DAC registers, followed by 480 lines of video data, each line containing 640 bytes (640 pixels). Image files must be unpacked using the PKUNZIP.EXE utility provided on the diskette before they can be displayed by SHOW.EXE.

GRAB.COM is a TSR program which can be used to save the contents of display memory to a file. To save an image, press the <SHIFT><PrtSc> keys. The first image will be saved with the filename PICTURE0.IMG, the second into PICTURE1.IMG, and so on. No compression is done on the image data.

# 7

# Programming Examples 256-Color Graphics

# Introduction

256-color modes are becoming increasingly popular because of the image quality that can be achieved. High resolution 256-color graphics modes are useful for applications such as presentation graphics where photographic image quality is warranted. As display memories continue to increase in size, these could become the most common modes for VGA programming. Drawing algorithms tend to be simple and efficient because bit masking and plane switching are not required. One byte of display memory is equivalent to one pixel on the screen.

Common 256-color resolutions are 640x400, which requires just 256K of display memory, 640x480 and 800x600 which require 512K of display memory, and 1024x768, which requires 1024K of display memory. Some drawing algorithms can be simplified for 1024x768 resolution, since a page boundary will never occur in mid-scanline.

The programming examples in this chapter illustrate programming techniques for these modes, using packed pixels with eight bits per pixel. They will show how to draw graphics primitives such as pixels, lines and rectangles, and how to perform BITBLT transfers. These examples are intended to be usable on any VGA board that supports any resolution in 256 colors and packed pixels. It is assumed that the board has already been initialized to the appropriate graphics mode before the programming examples are invoked.

# Display Memory Organization

Figure 7-1 shows the organization of display memory for these modes. Each pixel occupies one byte in the display memory. To convert from a pixel position, in X and Y coordinates, to page and offset in the display memory, using 64K pages, the following relations can be used:

| | |
|---|---|
| Page | = (Video_Pitch x Y + X)/10000h |
| Segment | = A000h |
| Byte Offset | = (Video_Pitch x Y + X) mod 10000h |

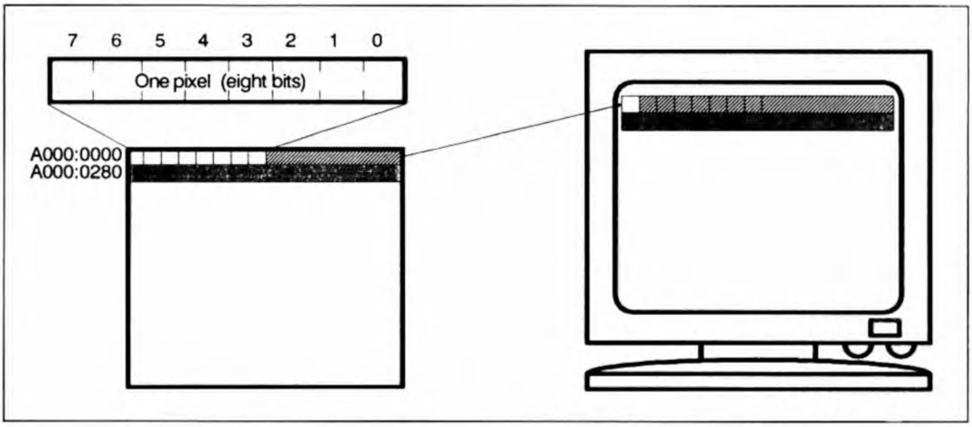


Figure 7-1. Display memory organization - 256-color graphics

# Drawing Routines

## Write Pixel

_Write_Pixel is a simple example that shows how to access a pixel with screen coordinates x,y. The x,y coordinate is used to compute a page and offset using the 16-bit multiply instruction MUL followed by a 32-bit add. At the completion of these two operations, register DX contains the page number and register AX contains the offset. Page selection is performed using the board-dependent procedure _Select_Page. The pixel can be accessed directly using the offset in AX and the mode-dependent segment variable Graf_Seg. No VGA registers need to be manipulated, and no masking operations are needed.

**Listing 7-1. File: 256COL\WPIXEL.ASM**

```
;************************************************************************
;*                                                                      *
;* File:        WPIXEL.ASM - 8 Bit Packed Pixel Write                   *
;* Routine:     _Write_Pixel                                            *
;* Arguments:   X, Y, Color                                             *
;*                                                                      *
;************************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pitch:WORD

        PUBLIC  _Write_Pixel

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_X           EQU     WORD PTR [BP+4]
```

```
Arg_Y           EQU     WORD PTR [BP+6]
Arg_Color       EQU     BYTE PTR [BP+8]

_Write_Pixel    PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Convert x,y pixel address to Page and Offset

        MOV     AX,Arg_Y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;   multiply by width in bytes
        ADD     AX,Arg_X                ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits
        MOV     DS,CS:Graf_Seg          ;Put new address in DS:SI
        MOV     DI,AX
        MOV     AL,DL                   ;Copy page number into AL
        CALL    Select_Page             ;Select proper page

        ; Set pixel to supplied value

        MOV     AL,Arg_Color            ;Fetch color to use
        MOV     [DI],AL                 ;Set the pixel

        ; Clean up and return

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Write_Pixel    ENDP

_TEXT   ENDS
        END
```

# Read Pixel

_Read_Pixel is a companion procedure to _Write_Pixel. In 256-color modes there are no substantial differences between the two procedures.

**Listing 7-2. File: 256COL\RPIXEL.ASM**

```
;*********************************************************************
;*                                                                   *
;* File:          RPIXEL.ASM - 8 Bit Packed Pixel Read               *
;* Routine:       _Read_Pixel                                        *
;* Arguments:     X, Y                                               *
;* Returns:       Color in AX                                        *
;*                                                                   *
;*********************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pitch:WORD

        PUBLIC  _Read_Pixel
```

```
_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_X           EQU     WORD PTR [BP+4]
Arg_Y           EQU     WORD PTR [BP+6]

_Read_Pixel     PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Convert x,y pixel address to Page and Offset

        MOV     AX,Arg_Y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;   multiply by width in bytes
        ADD     AX,Arg_X                ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits
        MOV     DS,CS:Graf_Seg          ;Put new address in DS:SI
        MOV     SI,AX
        MOV     AL,DL                   ;Copy page number into AL
        CALL    Select_Page             ;Select proper page

        ; Fetch the pixel value

        MOV     AL,[SI]                 ;Get byte of video memory
        XOR     AH,AH                   ;Clear upper byte (for return)

        ; Cleanup and return

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Read_Pixel     ENDP

_TEXT   ENDS
        END
```

## Draw Solid Line

_Line is used to demonstrate techniques used in incremental algorithms. An initial page and offset are computed from the starting x,y coordinate of the line. The line is then classified according to its slope (the relative size of DX and DY), and whether x and y are increasing or decreasing. Each line will fall into one of eight different classes, with different sections of code applying to each class.

Although some code sections could be combined to reduce total code size, the code is left in eight distinct sections to make it easier to add patterns and 'last pixel don't draw' checks. Each of the eight sections is divided into two parts: incremental drawing and page updating. For example, lines with positive DX and DY and DX greater than DY use the incremental drawing code between the labels NL_xp_yp and NL_fix0.

This code is a standard adaptation of Bresenham's line drawing algorithm, but with two added JC instructions for page boundary detection: one after x is updated (ADD DI,1) and one after y is updated (ADD DI,Pitch).

## Listing 7-3. File: 256COL\LINE.ASM

```
;*********************************************************************
;*                                                                   *
;* File:        LINE.ASM - 8 Bit Packed Solid Line                  *
;* Routine:     _Line(x0, y0, x1, y1, Color)                        *
;* Description: Draw line from (x0,y0) to (x1,y1) using color 'Color'   *
;*                                                                   *
;*********************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Video_Pitch:WORD
        EXTRN   Select_Page:NEAR

        PUBLIC  _Line

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x0          EQU     WORD PTR [BP+4] ;Formal parameters
Arg_y0          EQU     WORD PTR [BP+6]
Arg_x1          EQU     WORD PTR [BP+8]
Arg_y1          EQU     WORD PTR [BP+10]
Arg_Color       EQU     BYTE PTR [BP+12]

PageNo          EQU     BYTE PTR [BP-2] ;Local variables
D1              EQU     WORD PTR [BP-4] ;Local variables
D2              EQU     WORD PTR [BP-6]
Pitch           EQU     WORD PTR [BP-8]
Delta_x         EQU     WORD PTR [BP-10]
First_Mask      EQU     BYTE PTR [BP-12]

_Line           PROC NEAR
        PUSH    BP                      ;Standard C entry point
        MOV     BP,SP
        SUB     SP,12                   ;Declare local variables

        PUSH    DI                      ;Preserve segment registers
        PUSH    SI
        PUSH    DS
        PUSH    ES

;-----------------------------------------------------------------
; Convert (x,y) starting point to Seg:Off and select page
;-----------------------------------------------------------------

        MOV     AX,Arg_y0               ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;   multiply by width in bytes
        ADD     AX,Arg_x0               ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits

        PUSH    AX                      ;Save offset within page
        MOV     PageNo,DL               ;Save new page selection
        MOV     AL,DL                   ;Load page number into AL
        CALL    Select_Page             ;Select page
        MOV     DS,CS:Graf_Seg          ;Setup segment of address

;-----------------------------------------------------------------
; Compute dx and dy and determine which coordinate is major
;-----------------------------------------------------------------

        MOV     AX,CS:Video_Pitch       ; set raster increment
        MOV     Pitch,AX
        MOV     SI,Arg_x1               ; compute dx         reg-si
        SUB     SI,Arg_x0
        MOV     Delta_x,SI
        JGE     dxispos
        NEG     SI
dxispos:
```

```
        MOV     DI,Arg_y1               ; compute dy          reg-di
        SUB     DI,Arg_y0
        JGE     dyispos
        NEG     WORD PTR Pitch
        NEG     DI
dyispos:

        ; Determine which coordinate is the major one

        CMP     SI,DI                   ; check that dx > dy
        JGE     NL_xmajor
        JMP     NL_ymajor

;-----------------------------------------------------------------
; Diagonal line for x-major
;-----------------------------------------------------------------

NL_JumpToDone:
        POP     DI                      ;Restore stack
        JMP     NL_linedone

        ;-----------------------------------------------------------------
        ; Initialize error terms and updates for x-major
        ;-----------------------------------------------------------------

NL_xmajor:
        MOV     CX,SI                   ; set counter to dx+1
        INC     CX
        SAL     DI,1                    ; d1 = dy*2            reg-di
        MOV     DX,DI                   ; d  = dy*2-dx         reg-bx
        SUB     DX,SI
        NEG     SI                      ; d2 = dy*2-dx-dx      reg-si
        ADD     SI,DX
        MOV     d1,DI                   ; save d1
        MOV     d2,SI                   ; save d2
        POP     DI                      ; restore offset of first pixel

        ; Jump according to sign of dx and dy

        TEST    WORD PTR Pitch,8000h    ; Check if dy is positive
        JZ      NL_yp
        NEG     WORD PTR Pitch          ; Restore pitch
        TEST    WORD PTR Delta_x,8000h
        JNZ     NL_long_jump
        JMP     NL_xpyn                 ; go do dy negative dx positive
NL_long_jump:
        JMP     NL_xnyn                 ; go do both dy and dx negative

NL_yp:
        TEST    WORD PTR Delta_x,8000h  ; ...no, check if dx also positive
        JZ      NL_xpyp                 ; ...both dx and dy are positive
        JMP     NL_xnyp                 ; ...dx is negative and dy positive

        ;-----------------------------------------------------------------
        ; Draw line where DX > 0 and DY > 0 and x major
        ;-----------------------------------------------------------------

        ; Incremental drawing part

NL_xpyp:
        MOV     AL,Arg_Color            ; Fetch color of the pixel
NL_next0:                               ; Loop over pixels to be set
        MOV     [DI],AL                 ; Set next pixel
        ADD     DI,1                    ; Advance to next x
        JC      NL_fix0                 ; Go to select next page if needed
NL_page0:
        TEST    DX,8000h                ; if d >= 0 then ...
        JNZ     NL_dneg0
        ADD     DX,d2                   ; ... d = d + d2
        ADD     DI,Pitch                ; ... advance to next y
        JC      NL_fix00                ; ... go to select next page if needed
```

```
        LOOP    NL_next0
        JMP     NL_linedone

NL_dneg0:
        ADD     DX,d1                   ; if d < 0 then d = d + d1
        LOOP    NL_next0
        JMP     NL_linedone

        ; Page update part

NL_fix0:                                ; Advance to next page after x crossing
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        INC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        JMP     SHORT NL_page0

NL_fix00:                               ; Advance to next page after y crossing
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        INC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        LOOP    NL_next0
        JMP     NL_linedone

        ;-------------------------------------------------------------
        ; Draw line where DX < 0 and DY > 0 and X major
        ;-------------------------------------------------------------
NL_xnyp:
        MOV     AL,Arg_Color            ; Fetch color of the pixel
NL_next3:                               ; Loop over pixels to be set
        MOV     [DI],AL                 ; Set next pixel
        SUB     DI,1                    ; update offset
        JC      NL_fix3
NL_page3:
        TEST    DX,8000h                ; if d >= 0 then ...
        JNZ     NL_dneg3
        ADD     DX,d2                   ; ... d = d + d2
        ADD     DI,Pitch                ; update offset
        JC      NL_fix33
        LOOP    NL_next3
        JMP     NL_linedone

NL_dneg3:
        ADD     DX,d1                   ; if d < 0 then d = d + d1
        LOOP    NL_next3
        JMP     NL_linedone

NL_fix3:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        DEC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        JMP     SHORT NL_page3
NL_fix33:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        INC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        LOOP    NL_next3
        JMP     NL_linedone

        ;-------------------------------------------------------------
        ; Draw line where DX > 0 and DY < 0 and x major
        ;-------------------------------------------------------------
NL_xpyn:
        MOV     AL,Arg_Color            ; Fetch color of the pixel
NL_next7:                               ; Loop over pixels to be set
        MOV     [DI],AL                 ; Set next pixel
        ADD     DI,1                    ; update offset
        JC      NL_fix7
```

```
NL_page7:
        TEST    DX,8000h                ; if d >= 0 then ...
        JNZ     NL_dneg7
        ADD     DX,d2                   ; ... d = d + d2
        SUB     DI,Pitch                ; update offset
        JC      NL_fix77
        LOOP    NL_next7
        JMP     NL_linedone
NL_dneg7:
        ADD     DX,d1                   ; if d < 0 then d = d + d1
        LOOP    NL_next7
        JMP     NL_linedone

NL_fix7:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        INC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        JMP     SHORT NL_page7
NL_fix77:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        DEC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        LOOP    NL_next7
        JMP     NL_linedone

        ;------------------------------------------------------------
        ; Draw line where DX < 0 and DY < 0 and x major
        ;------------------------------------------------------------
NL_xnyn:
        MOV     AL,Arg_Color            ; Fetch color of the pixel
NL_next4:                               ; Loop over pixels to be set
        MOV     [DI],AL                 ; Set next pixel
        SUB     DI,1                    ; update offset
        JC      NL_fix4
NL_page4:
        TEST    DX,8000h                ; if d >= 0 then ...
        JNZ     NL_dneg4
        ADD     DX,d2                   ; ... d = d + d2
        SUB     DI,Pitch                ; update offset
        JC      NL_fix44
        LOOP    NL_next4
        JMP     NL_linedone
NL_dneg4:
        ADD     DX,d1                   ; if d < 0 then d = d + d1
        LOOP    NL_next4
        JMP     NL_linedone

NL_fix4:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        DEC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        JMP     SHORT NL_page4
NL_fix44:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        DEC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        LOOP    NL_next4
        JMP     NL_linedone

;------------------------------------------------------------
; Diagonal line for y-major
;------------------------------------------------------------
NL_JumpToDone1:
        POP     DI                      ;Restore stack
        JMP     NL_linedone

        ; Compute constants for dx < dy
```

```
NL_ymajor:
        MOV     CX,DI                   ; set counter to dy+1
        INC     CX
        SAL     SI,1                    ; d1 = dx * 2
        MOV     DX,SI                   ; d  = dx * 2 - dy
        SUB     DX,DI
        NEG     DI                      ; d2 = -dy + dx * 2 - dy
        ADD     DI,DX
        MOV     d2,DI                   ; save d2
        MOV     d1,SI                   ; save d1
        POP     DI                      ; Restore address of first pixel

        ; Jump according to sign of dx and dy

        TEST    WORD PTR Pitch,8000h    ; Check if dy is positive
        JZ      NL_py
        NEG     WORD PTR Pitch
        TEST    WORD PTR Delta_x,8000h
        JNZ     NL_jump_long
        JMP     NL_pxny                 ; go do dy negative dx positive
NL_jump_long:
        JMP     NL_nxny                 ; go do both dy and dx negative

NL_py:
        TEST    WORD PTR Delta_x,8000h  ; ...no, check if dx also positive
        JZ      NL_pxpy                 ; ...both dx and dy are positive
        JMP     NL_nxpy                 ; ...dx is negative and dy positive

        ;-------------------------------------------------------------
        ; Draw line where DX > 0 and DY > 0 and y major
        ;-------------------------------------------------------------

NL_pxpy:
        MOV     AL,Arg_Color            ; Fetch color of the pixel
NL_next1:
        MOV     [DI],AL                 ; Set next pixel
        ADD     DI,Pitch                ; update offset
        JC      NL_fix1
NL_page1:
        TEST    DX,8000h                ; if d >= 0 then
        JNZ     NL_dneg1

        ADD     DX,d2                   ; ... d = d + d2
        ADD     DI,1                    ; ... update offset
        JC      NL_fix11
        LOOP    NL_next1
        JMP     NL_linedone
NL_dneg1:
        ADD     DX,d1                   ; if d < 0 then d = d + d1
        LOOP    NL_next1
        JMP     NL_linedone

NL_fix1:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        INC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        JMP     SHORT NL_page1
NL_fix11:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        INC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        LOOP    NL_next1
        JMP     NL_linedone

        ;-------------------------------------------------------------
        ; Draw line where DX < 0 and DY > 0 and y major
        ;-------------------------------------------------------------

NL_nxpy:
```

```
        MOV     AL,Arg_Color            ; Fetch color of the pixel
NL_next2:
        MOV     [DI],AL                 ; Set next pixel
        ADD     DI,Pitch                ; update offset
        JC      NL_fix2
NL_page2:
        TEST    DX,8000h                ; if d >= 0 then
        JNZ     NL_dneg2

        ADD     DX,d2                       d = d + d2
        SUB     DI,1                        update offset
        JC      NL_fix22
        LOOP    NL_next2
        JMP     NL_linedone
NL_dneg2:
        ADD     DX,d1                   ; if d < 0 then d = d + d1
        LOOP    NL_next2
        JMP     NL_linedone

NL_fix2:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        INC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        JMP     SHORT NL_page2
NL_fix22:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        DEC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        LOOP    NL_next2
        JMP     NL_linedone

        ;-----------------------------------------------------------
        ; Draw line where DX > 0 and DY < 0 and y major
        ;-----------------------------------------------------------

NL_pxny:
        MOV     AL,Arg_Color            ; Fetch color of the pixel
NL_next6:
        MOV     [DI],AL                 ; Set next pixel
        SUB     DI,Pitch                ; update offset
        JC      NL_fix6
NL_page6:
        TEST    DX,8000h                ; if d >= 0 then ...
        JNZ     NL_dneg6

        ADD     DX,d2                   ; ... d = d + d2
        ADD     DI,1                    ; ... update offset
        JC      NL_fix66
        LOOP    NL_next6
        JMP     NL_linedone

NL_dneg6:
        ADD     DX,d1                   ; if d < 0 then d = d + d1
        LOOP    NL_next6
        JMP     NL_linedone

NL_fix6:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        DEC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        JMP     SHORT NL_page6
NL_fix66:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        INC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        LOOP    NL_next6
        JMP     NL_linedone
```

```
        ;-------------------------------------------------------------
        ; Draw line where DX < 0 and DY < 0 and y major
        ;-------------------------------------------------------------

NL_nxny:
        MOV     AL,Arg_Color            ; Fetch color of the pixel
NL_next5:
        MOV     [DI],AL                 ; Set next pixel
        SUB     DI,Pitch                ; update offset
        JC      NL_fix5
NL_page5:
        TEST    DX,8000h                ; if d >= 0 then
        JNZ     NL_dneg5

        ADD     DX,d2                       d = d + d2
        SUB     DI,1                        update offset
        JC      NL_fix55
        LOOP    NL_next5
        JMP     NL_linedone

NL_dneg5:
        ADD     DX,d1                   ; if d < 0 then d = d + d1
        LOOP    NL_next5
        JMP     NL_linedone

NL_fix5:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        DEC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        JMP     SHORT NL_page5
NL_fix55:
        XCHG    AL,PageNo               ; Preserve AL, and fetch page number
        DEC     AL                      ; Update page number
        CALL    Select_Page             ; Select new page number
        XCHG    AL,PageNo               ; Save updated page, restore AL
        LOOP    NL_next5
        JMP     NL_linedone
NL_linedone:

;-------------------------------------------------------------
; Clean up and return to caller
;-------------------------------------------------------------

End_Line:
        POP     ES                      ;Restore segment registers
        POP     DS
        POP     SI
        POP     DI

        MOV     SP,BP                   ;Standard C exit point
        POP     BP
        RET
_Line           ENDP

_TEXT   ENDS
        END
```

## Draw Scan Line

Scan line fill is a key building block in most fill algorithms. In the programming example _Scan_Line, the input coordinates are first ordered so that X0 < X1, and the starting coordinate X0,Y is translated to Page:Offset. Scan line drawing is then performed in two parts. First a check is made to see if a page boundary will be crossed by adding the starting offset (register DI) to the number of bytes to be filled (register CX).

The carry flag will be set if a page boundary is crossed, in which case the section of scan line contained in the first page will be drawn using STOS instructions, and the byte count adjusted, before the second display page is selected. In the second step, the rest of the scan line (or all of the scan line, if no page boundary was detected) is drawn.

**Listing 7-4. File: 256COL\SCANLINE.ASM**

```
;*********************************************************************
;*                                                                   *
;* File:        SCANLINE.ASM - 8 Bit Packed Scan Line                *
;* Routine:     _Scan_Line(x0, x1, y, Color)                         *
;* Description: Fill scanline 'y' with color 'Color' starting at 'x0' *
;*              and ending at 'x1'.                                  *
;*                                                                   *
;*********************************************************************

        INCLUDE VGA.INC

        EXTRN   Video_Pitch:WORD
        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR

        PUBLIC  _Scan_Line

_TEXT   SEGMENT BYTE PUBLIC 'CODE'
Arg_X0          EQU     WORD PTR [BP+4] ;Formal parameters
Arg_X1          EQU     WORD PTR [BP+6]
Arg_Y           EQU     WORD PTR [BP+8]
Arg_Color       EQU     BYTE PTR [BP+10]

_Scan_Line      PROC NEAR
        PUSH    BP
        MOV     BP,SP

        PUSH    DI
        PUSH    SI
        PUSH    DS
        PUSH    ES

        MOV     AX,Arg_X0               ;Make sure that x1 >=x0
        MOV     CX,Arg_X1
        CMP     CX,AX
        JGE     In_Order
        MOV     Arg_X0,CX
        MOV     Arg_X1,AX

;-----------------------------------------------------------
; Compute address of first pixel, and width of scan
;-----------------------------------------------------------

        ; Compute page number and select the page

In_Order:
        MOV     AX,Arg_Y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;  multiply by width in bytes
        ADD     AX,Arg_X0               ;  add x coordinate to compute offset
        ADC     DX,0                    ;  add overflow to upper 16 bits

        MOV     DI,AX                   ;Set offset
        MOV     ES,CS:Graf_Seg          ;Set segment
        MOV     AL,DL                   ;Copy page to AL
        CALL    Select_Page             ;Select the page

        MOV     CX,Arg_x1               ;Fetch x0
        SUB     CX,Arg_x0               ;Compute width
        INC     CX
```

```
        MOV     AL,Arg_Color            ;Fetch color
        MOV     AH,AL                   ;Duplicate color in AH

;-----------------------------------------------------------------
; Draw the scanline
;-----------------------------------------------------------------

        ;Fill first page if page boundary may be crossed
        MOV     BX,CX                   ; Check if within page
        ADD     BX,DI
        JNC     Scan_One_Page
        SUB     CX,BX                   ; Number of bytes to do in this page
        SHR     CX,1                    ; Adjust for move of words
        REP     STOSW                   ; Write new data
        ADC     CX,CX
        REP     STOSB
        MOV     CX,BX                   ; Number of bytes to do in next page
        XCHG    AL,DL                   ; Fetch page number, and preserve AL
        INC     AL                      ; Adjust page number
        CALL    Select_Page             ; Select next page
        XCHG    AL,DL                   ; Save updated page no., restore AL
        JCXZ    Scan_Done
        ;Fill second (or only page)
Scan_One_Page:
        SHR     CX,1                    ; Adjust for move of words
        REP     STOSW                   ; Write all words of data
        ADC     CX,CX                   ; Write the last odd byte of data
        REP     STOSB
Scan_Done:

;-----------------------------------------------------------------
; Cleanup and exit
;-----------------------------------------------------------------

End_Scan_Line:
        POP     ES                      ;Restore saved registers
        POP     DS
        POP     SI
        POP     DI
        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
_Scan_Line      ENDP

_TEXT   ENDS
        END
```

# Fill Solid Rectangle

Rectangles are the easiest figures to fill. _Solid_Rect uses the same algorithm described for _Scan_Line, except that the procedure is repeated for a specified number of scan lines with an appropriate page and offset update between successive scan lines.

**Listing 7-5. File: 256COL\RECT.ASM**

```
;*************************************************************************
;*                                                                       *
;* File:        RECT.ASM - 8 Bit Packed Solid Rectangle                  *
;* Routine:     _Solid_Rect(x0, y0, x1, y1, Color)                       *
;* Description: Draw a solid rectangle with corners at (x0,y0) and       *
;*              (x1,y1), filling the interior with color 'Color'         *
;*                                                                       *
;*************************************************************************
```

```
        INCLUDE VGA.INC

        EXTRN   Video_Pitch:WORD
        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR

        PUBLIC  _Solid_Rect

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_X0          EQU     WORD PTR [BP+4] ;Formal parameters
Arg_Y0          EQU     WORD PTR [BP+6]
Arg_X1          EQU     WORD PTR [BP+8]
Arg_Y1          EQU     WORD PTR [BP+10]
Arg_Color       EQU     BYTE PTR [BP+12]

PageNo          EQU     BYTE PTR [BP-2] ;Local variables
Pitch           EQU     WORD PTR [BP-4]

_Solid_Rect     PROC NEAR
        PUSH    BP
        MOV     BP,SP
        SUB     SP,4

        PUSH    DI                      ; Preserve registers
        PUSH    SI
        PUSH    DS
        PUSH    ES

;-----------------------------------------------------------------
; Rearrange corners so that x0 < x1 and y0 < y1
;-----------------------------------------------------------------

        MOV     AX,Arg_X0               ; Force x0 < x1
        MOV     BX,Arg_X1
        CMP     BX,AX
        JGE     X_In_Order
        MOV     Arg_X0,BX
        MOV     Arg_X1,AX
X_In_Order:
        MOV     AX,Arg_Y0               ; Force y0 < y1
        MOV     BX,Arg_Y1
        CMP     BX,AX
        JGE     Y_In_Order
        MOV     Arg_Y0,BX
        MOV     Arg_Y1,AX
Y_In_Order:

;-----------------------------------------------------------------
; Compute address of first pixel (upper left corner), and dimensions
;-----------------------------------------------------------------

        MOV     AX,Arg_y0               ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;   multiply by width in bytes
        ADD     AX,Arg_x0               ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits

        MOV     DI,AX                   ; Save offset within page
        MOV     PageNo,DL               ; Save new plane selection
        MOV     AL,DL                   ; Copy page number to AL
        CALL    Select_Page             ; Select page
        MOV     ES,CS:Graf_Seg          ; Set segment registers

        MOV     CX,Arg_x1               ; Set counter of bytes to do
        SUB     CX,Arg_x0               ; as (x2 - x1 + 1)
        INC     CX
        MOV     BX,CS:Video_Pitch       ; Compute pitch increment
        SUB     BX,CX
        MOV     Pitch,BX

        MOV     DX,Arg_y1               ; Set counter of rasters to do
```

```
        SUB     DX,Arg_y0               ; as (y2 - y1 + 1)
        INC     DX

        MOV     AL,Arg_Color            ; Fetch color
        MOV     AH,AL                   ; Duplicate color in both bytes

;--------------------------------------------------------------
; Fill the rectangle
;--------------------------------------------------------------

Scan_Loop:
        PUSH    CX                      ; Preserve byte counter

        ;Fill first page if page boundary may be crossed
        MOV     BX,CX                   ; Chech if within page
        ADD     BX,DI
        JNC     Scan_In_Page
        SUB     CX,BX                   ; Number of bytes to do in this page
        SHR     CX,1                    ; Adjust for move ofwords
        REP     STOSW                   ; Write new data
        ADC     CX,CX
        REP     STOSB
        MOV     CX,BX                   ; Number of bytes to do in next page
        XCHG    AL,PageNo               ; Fetch page number, and preserve AL
        INC     AL                      ; Adjust page number
        CALL    Select_Page             ; Select next page
        XCHG    AL,PageNo               ; Save updated page no., restore AL
        JCXZ    Scan_Done
        ;Fill second (or only page)
Scan_In_Page:
        SHR     CX,1                    ; Adjust for move of words
        REP     STOSW                   ; Write all words of data
        ADC     CX,CX                   ; Write the last odd byte of data
        REP     STOSB
Scan_Done:

        POP     CX                      ; Restore counter of bytes in a raster
        ADD     DI,Pitch                ; Compute ptr to byte in next raster
        JC      Rect_Fix_Page
        DEC     DX                      ; check if more rasters to do
        JG      Scan_Loop
        JMP     SHORT End_Rect

Rect_Fix_Page:
        XCHG    AL,PageNo               ; Fetch page number, and preserve AL
        INC     AL                      ; Update page number
        CALL    Select_Page             ; Compute and select new page number
        XCHG    AL,PageNo               ; Save updated page no., restore AL
        DEC     DX                      ; check if more rasters to do
        JG      Scan_Loop

;--------------------------------------------------------------
; Clean up and return to caller
;--------------------------------------------------------------

End_Rect:
        POP     ES                      ; Restore saved registers
        POP     DS
        POP     SI
        POP     DI
        MOV     SP,BP                   ; Restore stack
        POP     BP
        RET
_Solid_Rect     ENDP

_TEXT   ENDS
        END
```

# Clear Screen

A full screen can be filled most efficiently by avoiding all address translations and page boundary detection. _Clear_Screen shows how to efficiently erase the screen. At the start of the procedure, display refresh is disabled to allow data to be moved into display memory at the fastest possible rate. Display refresh normally imposes wait states on the processor when display memory is read or written. Display refresh is re-enabled at the end of the procedure.

**Listing 7-6. File: 256COL\CLEAR.ASM**

```
;**************************************************************************
;*                                                                        *
;* File:         CLEAR.ASM - 8 Bit Packed Pixel Clear Screen              *
;* Routine:      _Clear_Screen                                            *
;* Arguments:    Color                                                    *
;*                                                                        *
;**************************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pages:WORD

        PUBLIC  _Clear_Screen

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_Color       EQU     BYTE PTR [BP+4]

_Clear_Screen   PROC NEAR
        PUSH    BP                      ;Standard high-level entry
        MOV     BP,SP

        PUSH    ES                      ;Preserve registers
        PUSH    DI

        ; Enable maximum access to display memory (disable video refresh)

        MOV     DX,SEQUENCER_PORT       ;Fetch address of sequencer
        MOV     AL,1                    ;Index of clock select register
        OUT     DX,AL                   ;Select register
        INC     DX
        IN      AL,DX                   ;Read current value (to be restored)
        MOV     AH,AL                   ;Save current value
        OR      AL,20h                  ;Set disable video bit
        OUT     DX,AL                   ;Disable video refresh
        MOV     AL,1
        PUSH    AX                      ;Save old value for later

        ; Clear display memory

        XOR     BX,BX                   ;Initialize page counter
        MOV     AH,Arg_Color            ;Color to fill with
        MOV     AL,AH                   ;Duplicate in AH
        MOV     ES,CS:Graf_Seg          ;Select first segment

Cls_Page_Loop:
        XCHG    AL,BL                   ;Set page number in AL
        CALL    Select_Page             ;Select next page
        XCHG    AL,BL                   ;Restore fill color
        XOR     DI,DI                   ;Set offset
```

```
        MOV     CX,8000h                ;Number of words to clear
        REP     STOSW                   ;Clear the next segment
        INC     BX                      ;Update page counter
        CMP     BX,CS:Video_Pages       ;All pages cleared?
        JL      Cls_Page_Loop           ;If not go clear next one

        ; Restore video refresh

        MOV     DX,SEQUENCER_PORT       ;Fetch address of sequencer
        POP     AX                      ;Fetch previous value
        OUT     DX,AX                   ;Restore

        POP     DI                      ;Restore registers
        POP     ES
        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
_Clear_Screen   ENDP

_TEXT   ENDS
        END
```

# Copy Block

_BitBlt shows how to perform simple block copying where both the source and destination are in display memory. It is the only example in this text that utilizes the dual page capability of some VGA boards to improve performance. The module BITBLT.ASM is divided into three parts according to the capabilities of the board being used (as defined by the global variable Two_Pages). The organization of the module is as follows:

If two separate 32K read-write pages are not supported:

```
    {
    If no reversal needed:
          {
          Compute starting point for source & destination
          If source & destination in same page:
               Copy using MOVS, no page check needed.
          Else if separate read & write page not supported:
               Copy using intermediate buffer.
          Else if separate R & W, src & dst in different pages:
               Copy using MOVSB, check for page crossings.
          }
    Else if x reversal needed (because of overlap of source and destination):
          {
          Adjust starting point and counters,
          then proceed same as above (one of three methods).
          }
    Else if y reversal needed (because of overlap of source and destination):
          {
          Adjust starting point and counters,
          then proceed same as above (one of three methods).
          }
    }
Else if two separate 32K read-write pages supported:
    {
    Enable dual paging.
    If no reversal needed,
          {
          Compute starting point for source & destination.
          Copy one raster at a time using MOVS, checking for page crossings.
```

```
        }
    Else if x reversal needed (because of overlap):
        {
        Adjust starting point and counters.
        Copy one raster at a time using MOVS, checking for page crossing.
        }
    Else if y reversal needed (because of overlap):
        {
        Adjust starting point and counters.
        Copy one raster at a time using MOVS, checking for page crossing.
        }
}
```

Note that the transfer is performed by one of three code sections depending on the capability of the board. Section 1 uses an intermediate buffer to move data between display memory pages. Section 2 selects simultaneous separate read and write memory pages. Section 3 uses two fully independent 32K memory pages.

Within each section, transfers are classified in one of three classes depending on whether the source and destination rectangles overlap: (1) traverse x left-to-right and y top-to-bottom, (2) reverse x traversal, and (3) reverse y traversal.

For VGAs that allow only one page of display memory to be selected, a check is made to see if both source and destination lie within the same page. If not, an intermediate buffer in system memory must be used to transfer data between pages.

For each case, BITBLT data is transferred one scan line at a time using REP MOVS instructions if no page boundaries exist on the scan line. If a page boundary is detected on a scan line (for either the source or the destination), data is transferred one byte at a time.

### Listing 7-7. File: 256COL\BITBLT.ASM

```
;************************************************************************
;*                                                                      *
;* File:         BITBLT.ASM - 8 Bit Packed Bit Block Transfer           *
;* Routine:      _BitBlt                                                *
;* Arguments:    Source X, Source Y, Destination X, Destination Y,      *
;*               Width, Height                                          *
;*                                                                      *
;************************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Video_Pitch:WORD
        EXTRN   Ras_Buffer:BYTE
        EXTRN   Select_Page:NEAR
        EXTRN   Select_Read_Page:NEAR
        EXTRN   Select_Write_Page:NEAR
        EXTRN   Two_Pages:BYTE
        EXTRN   Enable_Dual_Page:NEAR
        EXTRN   Disable_Dual_Page:NEAR

        PUBLIC  _BitBlt

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_Src_X       EQU     WORD PTR [BP+4] ;Formal parameters
Arg_Src_Y       EQU     WORD PTR [BP+6]
Arg_Dst_X       EQU     WORD PTR [BP+8]
Arg_Dst_Y       EQU     WORD PTR [BP+10]
Arg_DX          EQU     WORD PTR [BP+12]
Arg_DY          EQU     WORD PTR [BP+14]

Pitch           EQU     WORD PTR [BP-02];Local variables
Src_Page        EQU     BYTE PTR [BP-04]
Dst_Page        EQU     BYTE PTR [BP-06]

_BitBlt PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        SUB     SP,6                    ;Allocate space for local variables

        PUSH    DS                      ;Preserve segment registers
        PUSH    ES
        PUSH    DI
        PUSH    SI

;-------------------------------------------------------------
; Check for support of two 32K pages (normally 64K pages are used)
;-------------------------------------------------------------

        TEST    CS:Two_Pages,02h        ;Check for two pages flag
        JZ      Not_Two_Pages           ;Do normal processing if 64k pages
        JMP     Two_32k_Pages           ;Go to 'faster' routines

;-------------------------------------------------------------
; Determine direction of traversal
;-------------------------------------------------------------

Not_Two_Pages:
        MOV     AX,Arg_Dst_Y
        CMP     AX,Arg_Src_Y
        JL      BB_XPYP                 ;If Y_DST above Y_SRC traverse
                                        ;    top to bottom
        JE      Check_X
        JMP     BB_XPYN                 ;If Y_DST below Y_SRC traverse
                                        ;    bottom to top
Check_X:                                ;IF Y_DST same as Y_SRC traverse
                                        ;    top to bottom and
```

```
        MOV     AX,Arg_Dst_X                revers x traversal if
        CMP     AX,Arg_Src_X                X_SRC to the left of X_DST
        JLE     BB_XPYP
        JMP     BB_XNYP

;-----------------------------------------------------------------
; Traverse x left-to-right and y top-to-bottom
;-----------------------------------------------------------------

BB_XPYP:
        ; Compute Page:Offset for first pixel in source and destination

        MOV     AX,Arg_Src_Y                ;Fetch y coordinate
        MUL     CS:Video_Pitch              ;   multiply by width in bytes
        ADD     AX,Arg_Src_X                ;   add x coordinate to compute offset
        ADC     DX,0                        ;   add overflow to upper 16 bits

        MOV     SI,AX                       ;Save the address
        MOV     Src_Page,DL
        MOV     AL,DL                       ;Select source page
        CALL    Select_Page
        MOV     DS,CS:Graf_Seg              ;Setup segment registers

        MOV     AX,CS:Video_Pitch           ; Compute row to row increment
        SUB     AX,Arg_DX
        MOV     Pitch,AX

        MOV     AX,Arg_Dst_Y                ;Fetch y coordinate
        MUL     CS:Video_Pitch              ;   multiply by width in bytes
        ADD     AX,Arg_Dst_X                ;   add x coordinate to compute offset
        ADC     DX,0                        ;   add overflow to upper 16 bits

        MOV     DI,AX                       ;Save address
        MOV     Dst_Page,DL
        MOV     ES,CS:Graf_Seg

        ; Check if both source and destination are within same page

        CMP     DL,Src_Page                 ;Are both src & dst in same page
        JNE     BB_Dif_Pages                ;...no, go do it the hard way

        MOV     AX,Arg_Src_Y                ;Compute address of last src pixel
        ADD     AX,Arg_DY
        DEC     AX
        MOV     CX,Arg_Src_X
        ADD     CX,Arg_DX
        DEC     CX
        MUL     CS:Video_Pitch                  multiply y by width in bytes
        ADD     AX,CX                           add x coordinate to compute offset
        ADC     DX,0                            add overflow to upper 16 bits

        CMP     DL,Src_Page                 ;Is last pixel in same page as first?
        JNE     BB_Dif_Pages                ;...no, go do it hard way
        MOV     AX,Arg_Dst_Y                ;Compute address of last dst pixel
        ADD     AX,Arg_DY
        DEC     AX
        MOV     CX,Arg_Dst_X
        ADD     CX,Arg_DX
        DEC     CX
        MUL     CS:Video_Pitch                  multiply y by width in bytes
        ADD     AX,CX                           add x coordinate to compute offset
        ADC     DX,0                            add overflow to upper 16 bits

        CMP     DL,Src_Page                 ;Is last pixels in same page as first?
        JNE     BB_Dif_Pages                ;...no, go do it the hard way

        ;-----------------------------------------------------------------
        ; Perform blit for src and dst in same page
        ;-----------------------------------------------------------------

BB_Same_Page:
```

```
BB_Line_Loop:
        MOV     CX,Arg_DX               ; Number of bytes to move
        SHR     CX,1
        REP     MOVSW
        ADC     CX,CX
        REP     MOVSB                   ; Move the bytes
        ADD     SI,Pitch                ; Update source pointer
        ADD     DI,Pitch                ; Update destination pointer
        DEC     Arg_DY
        JG      BB_Line_Loop            ; If not done go do move next row
        JMP     BB_Done

        ;-----------------------------------------------------------
        ; Perform blit from one page to another by
        ; copying one row at a time using an intermediate buffer
        ;-----------------------------------------------------------

BB_Dif_Pages:
        OR      CS:Two_Pages,0          ;Check if card capable of two pages
        JZ      BB_One_Page             ;...no, must use intermediate buffer
        JMP     BB_Two_Pages            ;...yes, go use two page pointers

        ; Move next row into temporary buffer
BB_One_Page:
        MOV     BX,DS                   ;Source segment
        MOV     DX,CS                   ;Destination segment
BB_Row:                                 ;Loop over rows to move
        MOV     CX,Arg_DX               ;Fetch block width
        PUSH    DI                      ;Preserve destination
        MOV     ES,DX                   ;Set ES to temporary buffer
        MOV     DS,BX                   ;Set DS to srouce segment
        LEA     DI,CS:Ras_Buffer        ;Use BX as index into tmp buffer
        MOV     AX,SI                   ;Check if source row in same page
        ADD     AX,CX
        JC      BB_Col                  ;...no, go do it one pixel at a time
        SHR     CX,1
        REP     MOVSW
        ADC     CX,CX
        REP     MOVSB                   ;...yes, do it the fast way
BB_Row_In:
        POP     DI
        ADD     SI,Pitch                ;Point to the next row
        JC      BB_Fix00
BB_00:
        MOV     AL,Dst_Page             ;Select destination page
        CALL    Select_Page

        ; Move next row into destination from temporary buffer
BB_Out:
        MOV     DS,DX                   ;Set DS to temporary buffer
        MOV     ES,BX                   ;Set ES to srouce segment
        MOV     CX,Arg_DX               ;Fetch block width
        PUSH    SI
        LEA     SI,CS:Ras_Buffer        ;Reset pointer within tmp buffer
        MOV     AX,DI                   ;Check if dest row in same page
        ADD     AX,CX
        JC      BB_Col1                 ;...no, go do it one pixel at a time
        SHR     CX,1
        REP     MOVSW
        ADC     CX,CX
        REP     MOVSB                   ;...yes, do it the fast way
BB_Row_Out:
        POP     SI
        ADD     DI,Pitch                ;Point to the next row
        JC      BB_Fix10
BB_10:
        MOV     AL,Src_Page             ;Select source page
        CALL    Select_Page
BB_11:
        DEC     Arg_DY                  ;Update counter of rows
        JG      BB_Row                  ;If not all rows done, go do another
```

```
        JMP     BB_Done

        ; Help code to move row across page boundary

BB_Col:                                 ;Loop over columns to move
        MOV     AL,[SI]                 ;Fetch source color
        STOSB                           ;Save in tmp buffer
        ADD     SI,1                    ;Update offset
        JC      BB_Fix20
        LOOP    BB_Col                  ;If not all bytes done, go do another
        JMP     BB_Row_In

BB_Col1:
        LODSB                           ;Fetch source byte
        MOV     ES:[DI],AL              ;save in destination
        ADD     DI,1                    ;update offset
        JC      BB_Fix30
        LOOP    BB_Col1                 ;If not all bytes done, go do another
        JMP     BB_Row_Out


        ; Help code to update and select next page

BB_Fix00:
        INC     Src_Page                ;Update page number
        JMP     BB_00

BB_Fix10:
        INC     Dst_Page                ;Update page number
        JMP     BB_10

BB_Fix20:
        INC     Src_Page                ;Update page number
        MOV     AL,Src_Page
        CALL    Select_Page             ;Compute and select new page number
        LOOP    BB_Col
        JMP     BB_Row_In

BB_Fix30:
        INC     Dst_Page                ;Update page number
        MOV     AL,Dst_Page
        CALL    Select_Page             ;Compute and select new page number
        LOOP    BB_Col1
        JMP     BB_Row_Out

        ;-------------------------------------------------------------
        ; Perform bitblt from one page to another, taking advantage
        ; of separate read and write pages
        ;-------------------------------------------------------------

BB_Two_Pages:
        MOV     DX,Arg_DY               ;Fetch block height
        MOV     AL,Src_Page             ;Select read page
        CALL    Select_Read_Page
        MOV     AL,Dst_Page             ;Select Write page
        CALL    Select_Write_Page

BB_Row2:
        MOV     CX,Arg_DX               ;Fetch block width
        MOV     AX,DI                   ;Check if dest row in same page
        ADD     AX,CX
        JC      BB_Col2                 ;...no, go do it one pixel at a time
        MOV     AX,SI                   ;Check if src row in same page
        ADD     AX,CX
        JC      BB_Col2                 ;...no, go do it one pixel at a time
        SHR     CX,1
        REP     MOVSW
        ADC     CX,CX
        REP     MOVSB                   ;...yes, do it the fast way
BB_Row_Done:
        ADD     SI,Pitch                ;Update source pointer to next row
```

```
        JC      BB_Fix40
BB_40:
        ADD     DI,Pitch                ;Update destination pointer to next row
        JC      BB_Fix50
BB_50:
        DEC     DX                      ;Update counter of rows
        JG      BB_Row2                 ;If not done go do next row
        JMP     BB_Done

        ; Help routines to fix page crossing

BB_Fix40:
        INC     Src_Page                ;Update page number
        MOV     AL,Src_Page
        CALL    Select_Read_Page        ;Compute and select new page number
        JMP     BB_40
BB_Fix50:
        INC     Dst_Page                ;Update page number
        MOV     AL,Dst_Page
        CALL    Select_Write_Page       ;Compute and select new page number
        JMP     BB_50
BB_Fix60:
        INC     Src_Page                ;Update read page
        MOV     AL,Src_Page
        CALL    Select_Read_Page
        JMP     BB_60
BB_Fix70:
        INC     Dst_Page                ;Update write page
        MOV     AL,Dst_Page
        CALL    Select_Write_Page
        LOOP    BB_Col2
        JMP     SHORT BB_Row_Done

        ; Help routines to copy a row accross a page boundary

BB_Col2:
        MOV     AL,DS:[SI]              ;Fetch source byte
        MOV     ES:[DI],AL              ;save in destination
        ADD     SI,1                    ;update offset
        JC      BB_Fix60
BB_60:
        ADD     DI,1
        JC      BB_Fix70
        LOOP    BB_Col2                 ;If not all bytes done, go do another
        JMP     SHORT BB_Row_Done

        ; Go to exit
BB_Done:
        JMP     End_BitBlt

;-------------------------------------------------------------
; Traverse x right-to-left and y from top-to-bottom
;-------------------------------------------------------------

BB_XNYP:
        ; Compute Page:Offset for first pixel in source and destination

        STD
        MOV     AX,Arg_Src_Y            ;Compute page and offset for source
        MOV     CX,Arg_Src_X
        ADD     CX,Arg_DX
        DEC     CX
        MUL     CS:Video_Pitch              multiply y by width in bytes
        ADD     AX,CX                       add x coordinate to compute offset
        ADC     DX,0                        add overflow to upper 16 bits

        MOV     SI,AX                   ;Save the address
        MOV     Src_Page,DL
        MOV     AL,DL                   ;Select source page
        CALL    Select_Page
        MOV     DS,CS:Graf_Seg          ;Setup segment registers
```

```
        MOV     AX,CS:Video_Pitch       ;Compute row to row increment
        ADD     AX,Arg_DX
        MOV     Pitch,AX

        MOV     AX,Arg_Dst_Y            ;Compute page and offset for dest
        MOV     CX,Arg_Dst_X
        ADD     CX,Arg_DX
        DEC     CX
        MUL     CS:Video_Pitch              multiply y by width in bytes
        ADD     AX,CX                       add x coordinate to compute offset
        ADC     DX,0                        add overflow to upper 16 bits

        MOV     DI,AX                   ;Save address
        MOV     Dst_Page,DL
        MOV     ES,CS:Graf_Seg

        ; Check if both source and destination are in same page

        CMP     DL,Src_Page             ;Are both src & dst in same page
        JNE     BBXR_Dif_Pages          ;...no, go do it the hard way

        MOV     AX,Arg_Src_Y            ;Compute address of last pixel
        ADD     AX,Arg_DY
        DEC     AX
        MUL     CS:Video_Pitch              multiply y by width in bytes
        ADD     AX,Arg_Src_X                add x coordinate to compute offset
        ADC     DX,0                        add overflow to upper 16 bits

        CMP     DL,Src_Page             ;Is last pixel in same page as first?
        JNE     BBXR_Dif_Pages          ;...no, go do it hard way
        MOV     AX,Arg_Dst_Y            ;Compute address of last pixel
        ADD     AX,Arg_DY
        DEC     AX
        MUL     CS:Video_Pitch              multiply y by width in bytes
        ADD     AX,Arg_Dst_X                add x coordinate to compute offset
        ADC     DX,0                        add overflow to upper 16 bits

        CMP     DL,Src_Page             ;Is last pixels in same page as first?
        JNE     BBXR_Dif_Pages          ;...no, go do it the hard way

        ;--------------------------------------------------------------
        ; Perform blit for src and dst in same page
        ;--------------------------------------------------------------

BBXR_Line_Loop:
        MOV     CX,Arg_DX               ; Number of bytes to move
        REP     MOVSB                   ; Move the bytes
        ADD     SI,Pitch                ; Update source pointer
        ADD     DI,Pitch                ; Update destination pointer
        DEC     Arg_DY
        JG      BBXR_Line_Loop          ; If not done go do move next row
        JMP     BBXR_Done

        ;--------------------------------------------------------------
        ; Perform blit from one page to another by
        ; copying one row at a time using an intermediate buffer
        ;--------------------------------------------------------------

BBXR_Dif_Pages:
        OR      CS:Two_Pages,0          ;Check if card capable of two pages
        JZ      BBXR_One_Page
        JMP     BBXR_Two_Pages          ;...yes, go use two page pointers

        ; Move next row into temporary buffer

BBXR_One_Page:
        MOV     BX,DS                   ;Source segment
        MOV     DX,CS                   ;Destination segment
BBXR_Row:                               ;Loop over rows to move
        MOV     CX,Arg_DX               ;Fetch block width
        PUSH    DI                      ;Preserve destination
```

```
        MOV     ES,DX                   ;Set ES to temporary buffer
        MOV     DS,BX                   ;Set DS to srouce segment
        LEA     DI,CS:Ras_Buffer[1023]  ;Use BX as index into tmp buffer
        MOV     AX,SI                   ;Check if source row in same page
        SUB     AX,CX
        JC      BBXR_Col                ;...no, go do it one pixel at a time
        REP     MOVSB                   ;...yes, do it the fast way
BBXR_Row_In:
        POP     DI
        ADD     SI,Pitch                ;Point to the next row
        JC      BBXR_Fix00
BBXR_00:
        MOV     AL,Dst_Page             ;Select destination page
        CALL    Select_Page

        ; Move next row into destination from temporary buffer
BBXR_Out:
        MOV     CX,Arg_DX               ;Fetch block width
        MOV     DS,DX                   ;Set DS to temporary buffer
        MOV     ES,BX                   ;Set ES to srouce segment
        PUSH    SI
        LEA     SI,CS:Ras_Buffer[1023]  ;Reset pointer within tmp buffer
        MOV     AX,DI                   ;Check if dest row in same page
        SUB     AX,CX
        JC      BBXR_Col1               ;...no, go do it one pixel at a time
        REP     MOVSB                   ;...yes, do it the fast way
BBXR_Row_Out:
        POP     SI
        ADD     DI,Pitch                ;Point to the next row
        JC      BBXR_Fix10
BBXR_10:
        MOV     AL,Src_Page             ;Select source page
        CALL    Select_Page
BBXR_11:
        DEC     Arg_DY                  ;Update counter of rows
        JG      BBXR_Row                ;If not all rows done, go do another
        JMP     BBXR_Done

        ; Help code to update and select next page

BBXR_Fix00:
        INC     Src_Page                ;Update page number
        JMP     BBXR_00

BBXR_Fix10:
        INC     Dst_Page                ;Update page number
        JMP     BBXR_10

BBXR_Fix20:
        DEC     Src_Page                ;Update page number
        MOV     AL,Src_Page
        CALL    Select_Page             ;Compute and select new page number
        LOOP    BBXR_Col
        JMP     BBXR_Row_In

BBXR_Fix30:
        DEC     Dst_Page                ;Update page number
        MOV     AL,Dst_Page
        CALL    Select_Page             ;Compute and select new page number
        LOOP    BBXR_Col1
        JMP     BBXR_Row_Out

        ; Help code to move row across page boundary

BBXR_Col:                               ;Loop over columns to move
        MOV     AL,[SI]                 ;Fetch source color
        STOSB                           ;Save in tmp buffer
        SUB     SI,1                    ;Update offset
        JC      BBXR_Fix20
        LOOP    BBXR_Col                ;If not all bytes done, go do another
        JMP     BBXR_Row_In
```

```
BBXR_Col1:
        LODSB                           ;Fetch source byte
        MOV     ES:[DI],AL              ;save in destination
        SUB     DI,1                    ;update offset
        JC      BBXR_Fix30
        LOOP    BBXR_Col1               ;If not all bytes done, go do another
        JMP     BBXR_Row_Out

        ;-----------------------------------------------------------------
        ; Perform bitblt from one page to another, taking advantage
        ; of separate read and write pages
        ;-----------------------------------------------------------------

BBXR_Two_Pages:
        MOV     DX,Arg_DY               ;Fetch height of the block
        MOV     AL,Src_Page             ;Select read page
        CALL    Select_Read_Page
        MOV     AL,Dst_Page             ;Select Write page
        CALL    Select_Write_Page

BBXR_Row2:
        MOV     CX,Arg_DX               ;Fetch block width
        MOV     AX,DI                   ;Check if dest row in same page
        SUB     AX,CX
        JC      BBXR_Col2               ; ...no, go do it one pixel at a time
        MOV     AX,SI                   ;Check if src row in same page
        SUB     AX,CX
        JC      BBXR_Col2               ; ...no, go do it one pixel at a time
        REP     MOVSB                   ;...yes, do it the fast way
BBXR_Row_Done:
        ADD     SI,Pitch                ;Update source pointer to next row
        JC      BBXR_Fix40
BBXR_40:
        ADD     DI,Pitch                ;Update destination pointer to next row
        JC      BBXR_Fix50
BBXR_50:
        DEC     DX                      ;Update counter of rows
        JG      BBXR_Row2               ;If not done go do next row

        JMP     BBXR_Done

        ; Help routines to fix page crossing

BBXR_Fix40:
        INC     Src_Page                ;Update page number
        MOV     AL,Src_Page
        CALL    Select_Read_Page        ;Compute and select new page number
        JMP     BBXR_40
BBXR_Fix50:
        INC     Dst_Page                ;Update page number
        MOV     AL,Dst_Page
        CALL    Select_Write_Page       ;Compute and select new page number
        JMP     BBXR_50
BBXR_Fix60:
        DEC     Src_Page                ;Update read page
        MOV     AL,Src_Page
        CALL    Select_Read_Page
        JMP     BBXR_60
BBXR_Fix70:
        DEC     Dst_Page                ;Update write page
        MOV     AL,Dst_Page
        CALL    Select_Write_Page
        LOOP    BBXR_Col2
        JMP     SHORT BBXR_Row_Done

        ; Help routines to copy a row accross a page boundary
BBXR_Col2:
        MOV     AL,DS:[SI]              ;Fetch source byte
        MOV     ES:[DI],AL              ;save in destination
        SUB     SI,1                    ;update offset
```

```
        JC      BBXR_Fix60
BBXR_60:
        SUB     DI,1
        JC      BBXR_Fix70
        LOOP    BBXR_Col2               ;If not all bytes done, go do another
        JMP     SHORT BBXR_Row_Done

        ; Go to exit

BBXR_Done:
        CLD                             ;Reset string direction
        JMP     End_BitBlt

;-----------------------------------------------------------------
; Traverse x left-to-right and y bottom-to-top
;-----------------------------------------------------------------

BB_XPYN:
        MOV     AX,Arg_Src_Y            ;Compute page and offset for source
        ADD     AX,Arg_DY
        DEC     AX
        MUL     CS:Video_Pitch              multiply y by width in bytes
        ADD     AX,Arg_Src_X                add x coordinate to compute offset
        ADC     DX,0                        add overflow to upper 16 bits

        MOV     SI,AX                   ;Save the address
        MOV     Src_Page,DL
        MOV     AL,DL                   ;Select source page
        CALL    Select_Page
        MOV     DS,CS:Graf_Seg          ;Setup segment registers
        MOV     AX,CS:Video_Pitch       ; Compute row to row increment
        ADD     AX,Arg_DX
        MOV     Pitch,AX

        MOV     AX,Arg_Dst_Y            ;Compute page and offset for dest
        ADD     AX,Arg_DY
        DEC     AX
        MUL     CS:Video_Pitch              multiply y by width in bytes
        ADD     AX,Arg_Dst_X                add x coordinate to compute offset
        ADC     DX,0                        add overflow to upper 16 bits

        MOV     DI,AX                   ;Save address
        MOV     Dst_Page,DL
        MOV     ES,CS:Graf_Seg

        ; Check if both source and destination are in same page

        CMP     DL,Src_Page             ;Are both src & dst  in same page
        JNE     BBRY_Dif_Pages          ;...no, go do it the hard way

        MOV     AX,Arg_Src_Y            ;Compute address of last pixel
        MOV     CX,Arg_Src_X
        ADD     CX,Arg_DX
        DEC     CX
        MUL     CS:Video_Pitch              multiply y by width in bytes
        ADD     AX,CX                       add x coordinate to compute offset
        ADC     DX,0                        add overflow to upper 16 bits

        CMP     DL,Src_Page             ;Is last pixel in same page as first?
        JNE     BBRY_Dif_Pages          ;...no, go do it hard way
        MOV     AX,Arg_Dst_Y            ;Compute address of last pixel
        MOV     CX,Arg_Dst_X
        ADD     CX,Arg_DX
        DEC     CX
        MUL     CS:Video_Pitch              multiply y by width in bytes
        ADD     AX,CX                       add x coordinate to compute offset
        ADC     DX,0                        add overflow to upper 16 bits

        CMP     DL,Src_Page             ;Is last pixels in same page as first?
        JNE     BBRY_Dif_Pages          ;...no, go do it the hard way
```

```
        ;-------------------------------------------------------------
        ; Perform blit for src and dst in same page
        ;-------------------------------------------------------------

BBRY_Line_Loop:
        MOV     CX,Arg_DX               ; Number of bytes to move
        SHR     CX,1
        REP     MOVSW                   ; Move the bytes
        ADC     CX,CX
        REP     MOVSB
        SUB     SI,Pitch                ; Update source pointer
        SUB     DI,Pitch                ; Update destination pointer
        DEC     Arg_DY
        JG      BBRY_Line_Loop          ; If not done go do move next row
        JMP     BBRY_Done

        ;-------------------------------------------------------------
        ; Perform blit from one page to another by
        ; copying one row at a time using an intermediate buffer
        ;-------------------------------------------------------------

BBRY_Dif_Pages:
        OR      CS:Two_Pages,0          ;Check if card capable of two pages
        JZ      BBRY_One_Page
        JMP     BBRY_Two_Pages          ;...yes, go use two page pointers

        ; Move next row into temporary buffer
BBRY_One_Page:
        MOV     BX,DS                   ;Source segment
        MOV     DX,CS                   ;Destination segment
BBRY_Row:                               ;Loop over rows to move
        MOV     CX,Arg_DX               ;Fetch width of the block
        PUSH    DI                      ;Preserve destination
        MOV     ES,DX                   ;Set ES to temporary buffer
        MOV     DS,BX                   ;Set DS to srouce segment
        LEA     DI,CS:Ras_Buffer        ;Use BX as index into tmp buffer
        MOV     AX,SI                   ;Check if source row in same page
        ADD     AX,CX
        JC      BBRY_Col                ;...no, go do it one pixel at a time
        SHR     CX,1
        REP     MOVSW                   ;...yes, do it the fast way
        ADC     CX,CX
        REP     MOVSB
BBRY_Row_In:
        POP     DI
        SUB     SI,Pitch                ;Point to the next row
        JC      BBRY_Fix00
BBRY_00:
        MOV     AL,Dst_Page             ;Select destination page
        CALL    Select_Page

        ; Move next row into destination from temporary buffer
BBRY_Out:
        MOV     CX,Arg_DX               ;Fetch width of the block
        MOV     DS,DX                   ;Set DS to temporary buffer
        MOV     ES,BX                   ;Set ES to srouce segment
        PUSH    SI
        LEA     SI,CS:Ras_Buffer        ;Reset pointer within tmp buffer
        MOV     AX,DI                   ;Check if dest row in same page
        ADD     AX,CX
        JC      BBRY_Col1               ;...no, go do it one pixel at a time
        SHR     CX,1
        REP     MOVSW                   ;...yes, do it the fast way
        ADC     CX,CX
        REP     MOVSB
BBRY_Row_Out:
        POP     SI
        SUB     DI,Pitch                ;Point to the next row
        JC      BBRY_Fix10
BBRY_10:
        MOV     AL,Src_Page             ;Select source page
```

```
        CALL    Select_Page
BBRY_11:
        DEC     Arg_DY                  ;Update counter of rows
        JG      BBRY_Row                ;If not all rows done, go do another
JMP     BBRY_Done

        ; Help code to move row across page boundary

BBRY_Col:                               ;Loop over columns to move
        MOV     AL,[SI]                 ;Fetch source color
        STOSB                           ;Save in tmp buffer
        ADD     SI,1                    ;Update offset
        JC      BBRY_Fix20
        LOOP    BBRY_Col                ;If not all bytes done, go do another
        JMP     BBRY_Row_In

BBRY_Col1:
        LODSB                           ;Fetch source byte
        MOV     ES:[DI],AL              ;save in destination
        ADD     DI,1                    ;update offset
        JC      BBRY_Fix30
        LOOP    BBRY_Col1               ;If not all bytes done, go do another
        JMP     BBRY_Row_Out

        ; Help code to update and select next page

BBRY_Fix00:
        DEC     Src_Page                ;Update page number
        JMP     BBRY_00

BBRY_Fix10:
        DEC     Dst_Page                ;Update page number
        JMP     BBRY_10

BBRY_Fix20:
        INC     Src_Page                ;Update page number
        MOV     AL,Src_Page
        CALL    Select_Page             ;Compute and select new page number
        LOOP    BBRY_Col
        JMP     BBRY_Row_In

BBRY_Fix30:
        INC     Dst_Page                ;Update page number
        MOV     AL,Dst_Page
        CALL    Select_Page             ;Compute and select new page number
        LOOP    BBRY_Col1
        JMP     BBRY_Row_Out

        ;------------------------------------------------------------
        ; Perform bitblt from one page to another, taking advantage
        ; of separate read and write pages
        ;------------------------------------------------------------

BBRY_Two_Pages:
        MOV     DX,Arg_DY               ;Fetch height of the block
        MOV     AL,Src_Page             ;Select read page
        CALL    Select_Read_Page
        MOV     AL,Dst_Page             ;Select Write page
        CALL    Select_Write_Page

BBRY_Row2:
        MOV     CX,Arg_DX               ;Fetch width of the block
        MOV     AX,DI                   ;Check if dest row in same page
        ADD     AX,CX
        JC      BBRY_Col2               ;...no, go do it one pixel at a time
        MOV     AX,SI                   ;Check if src row in same page
        ADD     AX,CX
        JC      BBRY_Col2               ;...no, go do it one pixel at a time
        SHR     CX,1
        REP     MOVSW
        ADC     CX,CX
```

```
        REP     MOVSB                   ;...yes, do it the fast way
BBRY_Row_Done:
        SUB     SI,Pitch                ;Update source pointer to next row
        JC      BBRY_Fix40
BBRY_40:
        SUB     DI,Pitch                ;Update destination pointer to next row
        JC      BBRY_Fix50
BBRY_50:
        DEC     DX                      ;Update counter of rows
        JG      BBRY_Row2               ;If not done go do next row
        JMP     BBRY_Done

        ; Help routines to fix page crossing

BBRY_Fix40:
        DEC     Src_Page                ;Update page number
        MOV     AL,Src_Page
        CALL    Select_Read_Page        ;Compute and select new page number
        JMP     BBRY_40
BBRY_Fix50:
        DEC     Dst_Page                ;Update page number
        MOV     AL,Dst_Page
        CALL    Select_Write_Page       ;Compute and select new page number
        JMP     BBRY_50
BBRY_Fix60:
        INC     Src_Page                ;Update read page
        MOV     AL,Src_Page
        CALL    Select_Read_Page
        JMP     BBRY_60
BBRY_Fix70:
        INC     Dst_Page                ;Update write page
        MOV     AL,Dst_Page
        CALL    Select_Write_Page
        LOOP    BBRY_Col2
        JMP     SHORT BBRY_Row_Done

        ; Help routines to copy a row accross a page boundary
BBRY_Col2:
        MOV     AL,DS:[SI]              ;Fetch source byte
        MOV     ES:[DI],AL              ;save in destination
        ADD     SI,1                    ;update offset
        JC      BBRY_Fix60
BBRY_60:
        ADD     DI,1
        JC      BBRY_Fix70
        LOOP    BBRY_Col2               ;If not all bytes done, go do another
        JMP     SHORT BBRY_Row_Done

        ; Go to exit
BBRY_Done:


;-------------------------------------------------------------
; Cleanup and return
;-------------------------------------------------------------

End_BitBlt:
        POP     SI                      ;Restore segment registers
        POP     DI
        POP     ES
        POP     DS
        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
_BitBlt ENDP
```

```
;-------------------------------------------------------------------
;-------------------------------------------------------------------
;-------------------------------------------------------------------
;-------   Routines to perform bitblt in two 32k pages ----------
;-------------------------------------------------------------------
;-------------------------------------------------------------------
;-------------------------------------------------------------------

;-------------------------------------------------------------------
; Set segment registers and enable dual paging
;-------------------------------------------------------------------

Two_32k_Pages:
        MOV     DS,CS:Graf_Seg[2]       ;Set segments for transfer
        MOV     ES,CS:Graf_Seg[0]
        CALL    Enable_Dual_Page        ;Eanble dual page paging

;-------------------------------------------------------------------
; Determine direction of traversal
;-------------------------------------------------------------------

        MOV     AX,Arg_Dst_Y
        CMP     AX,Arg_Src_Y
        JL      BB2_XPYP                ;If Y_DST above Y_SRC traverse
                                        ;    top to bottom
        JE      X_Check
        JMP     BB2_XPYN                ;If Y_DST below Y_SRC traverse
                                        ;    bottom to top
X_Check:                                ;IF Y_DST same as Y_SRC traverse
                                        ;    top to bottom and
        MOV     AX,Arg_Dst_X            ;    revers x traversal if
        CMP     AX,Arg_Src_X            ;    X_SRC to the left of X_DST
        JLE     BB2_XPYP
        JMP     BB2_XNYP

;-------------------------------------------------------------------
; Traverse x left-to-right and y top-to-bottom
;-------------------------------------------------------------------

BB2_XPYP:
        ; Compute Page:Offset for first pixel in source and destination

        MOV     AX,Arg_Src_Y            ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;   multiply by width in bytes
        ADD     AX,Arg_Src_X            ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits
        SHL     AX,1                    ;Convert 64k page number to 32k
        RCL     DX,1
        SHR     AX,1
        MOV     SI,AX                   ;Save the address
        MOV     Src_Page,DL

        MOV     AX,Arg_Dst_Y            ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;   multiply by width in bytes
        ADD     AX,Arg_Dst_X            ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits
        SHL     AX,1                    ;Convert 64k page number to 32k
        RCL     DX,1
        SHR     AX,1
        MOV     DI,AX                   ;Save address
        MOV     Dst_Page,DL

        MOV     AX,CS:Video_Pitch       ; Compute row to row increment
        SUB     AX,Arg_DX
        MOV     Pitch,AX

        ;-----------------------------------------------------------------
        ; Perform bitblt from one page to another, taking advantage
        ; of separate read and write pages
        ;-----------------------------------------------------------------
```

```
        MOV     DX,Arg_DY               ;Fetch block height
        MOV     AL,Src_Page             ;Select read page
        CALL    Select_Read_Page
        MOV     AL,Dst_Page             ;Select Write page
        CALL    Select_Write_Page

BB2_Row2:
        MOV     CX,Arg_DX               ;Fetch block width
        MOV     AX,DI                   ;Check if dest row in same page
        ADD     AX,CX
        JS      BB2_Col2                ;...no, go do it one pixel at a time
        MOV     AX,SI                   ;Check if src row in same page
        ADD     AX,CX
        JS      BB2_Col2                ;...no, go do it one pixel at a time
        SHR     CX,1
        REP     MOVSW
        ADC     CX,CX
        REP     MOVSB                   ;...yes, do it the fast way
BB2_Row_Done:
        ADD     SI,Pitch                ;Update source pointer to next row
        JS      BB2_Fix40
BB2_40:
        ADD     DI,Pitch                ;Update destination pointer to next row
        JS      BB2_Fix50
BB2_50:
        DEC     DX                      ;Update counter of rows
        JG      BB2_Row2                ;If not done go do next row
        JMP     BB2_Done

        ; Help routines to fix page crossing

BB2_Fix40:
        AND     SI,NOT 8000h            ;Clear sign bit
        INC     Src_Page                ;Update page number
        MOV     AL,Src_Page
        CALL    Select_Read_Page        ;Compute and select new page number
        JMP     BB2_40
BB2_Fix50:
        AND     DI,NOT 8000h            ;Clear sign bit
        INC     Dst_Page                ;Update page number
        MOV     AL,Dst_Page
        CALL    Select_Write_Page       ;Compute and select new page number
        JMP     BB2_50
BB2_Fix60:
        AND     SI,NOT 8000h            ;Clear sign bit
        INC     Src_Page                ;Update read page
        MOV     AL,Src_Page
        CALL    Select_Read_Page
        JMP     BB2_60
BB2_Fix70:
        AND     DI,NOT 8000h            ;Clear sign bit
        INC     Dst_Page                ;Update write page
        MOV     AL,Dst_Page
        CALL    Select_Write_Page
        LOOP    BB2_Col2
        JMP     SHORT BB2_Row_Done

        ; Help routines to copy a row accross a page boundary

BB2_Col2:
        MOV     AL,DS:[SI]              ;Fetch source byte
        MOV     ES:[DI],AL              ;save in destination
        ADD     SI,1                    ;update offset
        JS      BB2_Fix60
BB2_60:
        ADD     DI,1
        JS      BB2_Fix70
        LOOP    BB2_Col2                ;If not all bytes done, go do another
        JMP     SHORT BB2_Row_Done

        ; Go to exit
```

```
BB2_Done:
        JMP     End_2page_Blit

;------------------------------------------------------------
; Traverse x right-to-left and y from top-to-bottom
;------------------------------------------------------------

BB2_XNYP:
        ; Compute Page:Offset for first pixel in source and destination

        STD
        MOV     AX,Arg_Src_Y            ;Compute page and offset for source
        MOV     CX,Arg_Src_X
        ADD     CX,Arg_DX
        DEC     CX
        MUL     CS:Video_Pitch          ;   multiply y by width in bytes
        ADD     AX,CX                   ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits
        SHL     AX,1                    ;Convert 64k page number to 32k
        RCL     DX,1
        SHR     AX,1
        MOV     SI,AX                   ;Save the address
        MOV     Src_Page,DL

        MOV     AX,Arg_Dst_Y            ;Compute page and offset for dest
        MOV     CX,Arg_Dst_X
        ADD     CX,Arg_DX
        DEC     CX
        MUL     CS:Video_Pitch          ;   multiply y by width in bytes
        ADD     AX,CX                   ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits
        SHL     AX,1                    ;Convert 64k page number to 32k
        RCL     DX,1
        SHR     AX,1
        MOV     DI,AX                   ;Save address
        MOV     Dst_Page,DL

        MOV     AX,CS:Video_Pitch       ;Compute row to row increment
        ADD     AX,Arg_DX
        MOV     Pitch,AX

        ;------------------------------------------------------------
        ; Perform bitblt from one page to another, taking advantage
        ; of separate read and write pages
        ;------------------------------------------------------------

        MOV     DX,Arg_DY               ;Fetch height of the block
        MOV     AL,Src_Page             ;Select read page
        CALL    Select_Read_Page
        MOV     AL,Dst_Page             ;Select Write page
        CALL    Select_Write_Page

BB2XR_Row2:
        MOV     CX,Arg_DX               ;Fetch block width
        MOV     AX,DI                   ;Check if dest row in same page
        SUB     AX,CX
        JS      BB2XR_Col2              ; ...no, go do it one pixel at a time
        MOV     AX,SI                   ;Check if src row in same page
        SUB     AX,CX
        JS      BB2XR_Col2              ; ...no, go do it one pixel at a time
        REP     MOVSB                   ;...yes, do it the fast way
BB2XR_Row_Done:
        ADD     SI,Pitch                ;Update source pointer to next row
        JS      BB2XR_Fix40
BB2XR_40:
        ADD     DI,Pitch                ;Update destination pointer to next row
        JS      BB2XR_Fix50
BB2XR_50:
        DEC     DX                      ;Update counter of rows
        JG      BB2XR_Row2              ;If not done go do next row
```

```
        JMP     BB2XR_Done

        ; Help routines to fix page crossing

BB2XR_Fix40:
        AND     SI,NOT 8000h            ;Clear sign bit
        INC     Src_Page                ;Update page number
        MOV     AL,Src_Page
        CALL    Select_Read_Page        ;Compute and select new page number
        JMP     BB2XR_40
BB2XR_Fix50:
        AND     DI,NOT 8000h            ;Clear sign bit
        INC     Dst_Page                ;Update page number
        MOV     AL,Dst_Page
        CALL    Select_Write_Page       ;Compute and select new page number
        JMP     BB2XR_50
BB2XR_Fix60:
        AND     SI,NOT 8000h            ;Clear sign bit
        DEC     Src_Page                ;Update read page
        MOV     AL,Src_Page
        CALL    Select_Read_Page
        JMP     BB2XR_60
BB2XR_Fix70:
        AND     DI,NOT 8000h            ;Clear sign bit
        DEC     Dst_Page                ;Update write page
        MOV     AL,Dst_Page
        CALL    Select_Write_Page
        LOOP    BB2XR_Col2
        JMP     SHORT BB2XR_Row_Done

        ; Help routines to copy a row accross a page boundary
BB2XR_Col2:
        MOV     AL,DS:[SI]              ;Fetch source byte
        MOV     ES:[DI],AL              ;save in destination
        SUB     SI,1                    ;update offset
        JS      BB2XR_Fix60
BB2XR_60:
        SUB     DI,1
        JS      BB2XR_Fix70
        LOOP    BB2XR_Col2              ;If not all bytes done, go do another
        JMP     SHORT BB2XR_Row_Done

        ; Go to exit

BB2XR_Done:
        CLD                             ;Reset string direction
        JMP     End_2page_Blit

;------------------------------------------------------------
; Traverse x left-to-right and y bottom-to-top
;------------------------------------------------------------

BB2_XPYN:
        MOV     AX,Arg_Src_Y            ;Compute page and offset for source
        ADD     AX,Arg_DY
        DEC     AX
        MUL     CS:Video_Pitch          ;   multiply y by width in bytes
        ADD     AX,Arg_Src_X            ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits
        SHL     AX,1                    ;Convert 64k page number to 32k
        RCL     DX,1
        SHR     AX,1
        MOV     SI,AX                   ;Save the address
        MOV     Src_Page,DL

        MOV     AX,Arg_Dst_Y            ;Compute page and offset for dest
        ADD     AX,Arg_DY
        DEC     AX
        MUL     CS:Video_Pitch              multiply y by width in bytes
        ADD     AX,Arg_Dst_X                add x coordinate to compute offset
        ADC     DX,0                        add overflow to upper 16 bits
```

```
        SHL     AX,1                    ;Convert 64k page number to 32k
        RCL     DX,1
        SHR     AX,1
        MOV     DI,AX                   ;Save address
        MOV     Dst_Page,DL

        MOV     AX,CS:Video_Pitch       ; Compute row to row increment
        ADD     AX,Arg_DX
        MOV     Pitch,AX

        ;-----------------------------------------------------------------
        ; Perform bitblt from one page to another, taking advantage
        ; of separate read and write pages
        ;-----------------------------------------------------------------

        MOV     DX,Arg_DY               ;Fetch height of the block
        MOV     AL,Src_Page             ;Select read page
        CALL    Select_Read_Page
        MOV     AL,Dst_Page             ;Select Write page
        CALL    Select_Write_Page

BB2RY_Row2:
        MOV     CX,Arg_DX               ;Fetch width of the block
        MOV     AX,DI                   ;Check if dest row in same page
        ADD     AX,CX
        JS      BB2RY_Col2              ;...no, go do it one pixel at a time
        MOV     AX,SI                   ;Check if src row in same page
        ADD     AX,CX
        JS      BB2RY_Col2              ;...no, go do it one pixel at a time
        SHR     CX,1
        REP     MOVSW
        ADC     CX,CX
        REP     MOVSB                   ;...yes, do it the fast way
BB2RY_Row_Done:
        SUB     SI,Pitch                ;Update source pointer to next row
        JS      BB2RY_Fix40
BB2RY_40:
        SUB     DI,Pitch                ;Update destination pointer to next row
        JS      BB2RY_Fix50
BB2RY_50:
        DEC     DX                      ;Update counter of rows
        JG      BB2RY_Row2              ;If not done go do next row
        JMP     BB2RY_Done

        ; Help routines to fix page crossing

BB2RY_Fix40:
        AND     SI,NOT 8000h            ;Clear sign bit
        DEC     Src_Page                ;Update page number
        MOV     AL,Src_Page
        CALL    Select_Read_Page        ;Compute and select new page number
        JMP     BB2RY_40
BB2RY_Fix50:
        AND     DI,NOT 8000h            ;Clear sign bit
        DEC     Dst_Page                ;Update page number
        MOV     AL,Dst_Page
        CALL    Select_Write_Page       ;Compute and select new page number
        JMP     BB2RY_50
BB2RY_Fix60:
        AND     SI,NOT 8000h            ;Clear sign bit
        INC     Src_Page                ;Update read page
        MOV     AL,Src_Page
        CALL    Select_Read_Page
        JMP     BB2RY_60
BB2RY_Fix70:
        AND     DI,NOT 8000h            ;Clear sign bit
        INC     Dst_Page                ;Update write page
        MOV     AL,Dst_Page
        CALL    Select_Write_Page
        LOOP    BB2RY_Col2
        JMP     SHORT BB2RY_Row_Done
```

```
        ; Help routines to copy a row accross a page boundary
BB2RY_Col2:
        MOV     AL,DS:[SI]              ;Fetch source byte
        MOV     ES:[DI],AL              ;save in destination
        ADD     SI,1                    ;update offset
        JS      BB2RY_Fix60
BB2RY_60:
        ADD     DI,1
        JS      BB2RY_Fix70
        LOOP    BB2RY_Col2              ;If not all bytes done, go do another
        JMP     SHORT BB2RY_Row_Done

        ; Go to exit
BB2RY_Done:

;---------------------------------------------------------------
; Cleanup and return
;---------------------------------------------------------------

End_2page_Blit:
        CALL    Disable_Dual_Page       ;Disable dual page paging
        JMP     End_BitBlt

_TEXT   ENDS
        END
```

## Set Cursor, Move Cursor, Remove Cursor

This module contains three procedures to define, move, and remove a cursor in the display memory.

In the procedure _Set_Cursor, monochrome XOR and AND masks are expanded according to the parameters FG_Color (foreground color) and BG_Color (background color). In this implementation these masks are stored on screen in an area immediately below the first scan line in order to clearly show how the cursor is constructed. By changing one line of marked code, the cursor mask storage area can be moved off screen. The entire cursor mask storage area must reside within one page of display memory.

At the end of the _Set_Cursor procedure, the variables Last_Cursor_X and Last_Cursor_Y are initialized to ensure proper operation during first call to _Move_Cursor.

In the procedure _Move_Cursor, the cursor masks are logically combined with the background data from the new cursor position specified. A block twice the size of the cursor is used to minimize flicker for small changes in cursor position. Background data for a block around the cursor position is kept immediately next to the cursor masks. A check is made to see if the cursor moved outside of the current block, and if so, the cursor is removed from the screen (by calling _Remove_Cursor) and a new block is copied to the save area. Next, the background save area is copied into the build area (next to the save area), where the cursor masks are combined with the background data. The data in the build area is then copied to the display.

For a small motion of the cursor (within the same block), the cursor in the display area is removed and placed in its new position in a single transfer; the cursor never disappears from the screen and flicker is eliminated (until an edge of the block is reached).

_Remove_Cursor restores the area under the cursor by transferring data from the save area to the display.

With many VGAs the off-screen memory is not easily accessible when 256K modes are used. Mode select will not enable the second bank of 256K. For these boards, all of display memory can enabled in the routine _Select_Graphics in the board-dependent module SELECT.ASM.

### Listing 7-8. File: 256COL\CURSOR.ASM

```
;*********************************************************************
;*                                                                   *
;* File:  CURSOR.ASM - 8 Bit Packed Cursor Routines                  *
;* Description: Cursor manipulation routines                         *
;*              _Set_Cursor                                          *
;*              _Move_Cursor                                         *
;*              _Remove_Cursor                                       *
;*                                                                   *
;*********************************************************************

     INCLUDE VGA.INC

     EXTRN     Graf_Seg:WORD
     EXTRN     Video_Pitch:WORD
     EXTRN     Video_Height:WORD
     EXTRN     _BitBlt:NEAR
     EXTRN     Select_Page:NEAR

     PUBLIC    _Set_Cursor
     PUBLIC    _Move_Cursor
     PUBLIC    _Remove_Cursor

_TEXT     SEGMENT BYTE PUBLIC 'CODE'

;--------------------------------------------------------------------
; Common cursor definitions
;--------------------------------------------------------------------

CUR_WIDTH      EQU  32
CUR_HEIGHT     EQU  32

AND_OFFSET     EQU  0         ;Save area offsets in off-screen area
XOR_OFFSET     EQU  CUR_WIDTH
CUR_OFFSET     EQU  2*CUR_WIDTH
MIX_OFFSET     EQU  4*CUR_WIDTH

Last_Cursor_x  DW   0         ;Code segment variables
Last_Cursor_y  DW   0
Save_Area_y    DW   0
Save_Offset    DW   0
```

```
;***********************************************************************
;*                                                                     *
;* _Set_Cursor(AND_Mask, XOR_Mask, FG_Color, BG_Color)                 *
;*   This procedure will expand the two cursor masks into              *
;*   color.    Normally the masks should be stored after the           *
;*   last visible scan line (global parameter 'Video_Height',          *
;*   however in this demo, the cursor masks and the 'save buffer'      *
;*   will be stored immediately above the last line.  This is done     *
;*   so that the reader can clearly see the AND mask, the XOR mask,    *
;*   and the area under the cursor in 'save buffer'.                   *
;*                                                                     *
;* Entry:                                                              *
;*   AND_Mask - 4x32 bytes with AND mask                               *
;*   XOR_Mask - 4x32 bytes with XOR mask                               *
;*   BG_Color - Foreground color                                       *
;*   FG_Color - Background color                                       *
;*                                                                     *
;***********************************************************************

Arg_AND_Mask    EQU  WORD PTR [BP+4] ;Formal parameters
Arg_XOR_Mask    EQU  WORD PTR [BP+6]
Arg_BG_Color    EQU  BYTE PTR [BP+8]
Arg_FG_Color    EQU  BYTE PTR [BP+10]

_Set_Cursor     PROC NEAR
     PUSH BP              ;Standard high-level entry
     MOV  BP,SP
     SUB  SP,2

     PUSH SI              ;Save registers
     PUSH DI
     PUSH ES
     PUSH DS

     ; Fill with background

     MOV  CX,0            ;Set x to start of save area
     MOV  AX,CS:Video_Height  ;Set y to below last line on the screen
     ;!!!!!!!!!!!! The next line should be removed    !!!!!!!!!!!!!!!!!!!
     ;!!!!!!!!!!!! if you do not want to see the save !!!!!!!!!!!!!!!!!!!
     ;!!!!!!!!!!!! regions on the screen              !!!!!!!!!!!!!!!!!!!
     MOV  AX,0                        ;Make visible for demo !!!!!!!!!!!!!
     MOV  CS:Save_Area_y,AX    ;Save y for other cursor procs
     MUL  CS:Video_Pitch      ;   multiply y by width in bytes
     ADD  AX,CX               ;   add x coordinate to compute offset
     ADC  DX,0               add overflow to upper 16 bits

     MOV  DI,AX               ;Set DI to save area offset
     MOV  CS:Save_Offset,AX   ;Save offset for later
     MOV  ES,CS:Graf_Seg      ;Set segment to graphics segment
     MOV  AL,DL               ;Copy page number into AL
     CALL Select_Page         ;Select page for save area

     MOV  DX,CUR_HEIGHT       ;Number of scanlines to do
     MOV  BX,CS:Video_Pitch   ;Calculate scan-to-scan increment
     SUB  BX,CUR_WIDTH*2

     MOV  AL,Arg_BG_Color     ;Fetch background color
     MOV  AH,AL               ;Copy color into AH
Fill_Background:
     MOV  CX,CUR_WIDTH        ;Number of words of AND & XOR mask
     REP  STOSW               ;Fill next row of AND and XOR masks
     ADD  DI,BX               ;Point to next scanline (assumes in
                              ;one page!!!).
     DEC  DX                  ;Check if all scanlines done
     JG   Fill_Background     ;Go do next scanline if not done

     ; Change foreground bits for the AND mask save area
```

```
        MOV   DL,CUR_HEIGHT         ;Initialize raster counter
        MOV   DH,Arg_FG_Color       ;Fetch foreground color
        MOV   DI,CS:Save_Offset     ;Get pointer to save area
        MOV   SI,Arg_AND_Mask       ;Fetch pointer to AND-mask section
        ADD   BX,CUR_WIDTH          ;Adjust scan-to-scan increment

Set_AND_FG:
        LODSW                       ;Fetch next 16 bits from the mask
        XCHG AL,AH                  ;Swap byte to compensate for 80xx mem
        MOV   CX,16                 ;Number of bits to do
AND_Bit_Loop:
        SHL   AX,1                  ;Move next bit into carry
        JNC   AND_Done              ;Do not change if bit not set
        MOV   ES:[DI],DH            ;Set pixel to fg color if bit set
AND_Done:
        INC   DI                    ;Update pointer
        LOOP AND_Bit_Loop           ;If not all 16 bits done do next bit
        XOR   BX,8000h              ;Toggle high bit of BX to check if
        JS    Set_AND_FG            ; both words have been done

        ADD   DI,BX                 ;Point to next scanline
        DEC   DL                    ;Check if all scanlines done
        JG    Set_AND_FG            ;Go do next scanline if not done

        ; Change foreground bits for the XOR mask save area

        MOV   DL,CUR_HEIGHT         ;Initialize raster counter
        MOV   DH,Arg_FG_Color       ;Fetch foreground color
        MOV   DI,CS:Save_Offset     ;Get pointer to save area
        ADD   DI,XOR_OFFSET         ;Advance pointer to XOR-mask section
        MOV   SI,Arg_XOR_Mask       ;Fetch pointer to XOR-mask

Set_XOR_FG:
        LODSW                       ;Fetch next 16 bits from the mask
        XCHG AL,AH                  ;Swap byte to compensate for 80xx mem
        MOV   CX,16                 ;Number of bits to do
XOR_Bit_Loop:
        SHL   AX,1                  ;Move next bit into carry
        JNC   XOR_Done              ;Do not change if bit not set
        MOV   ES:[DI],DH            ;Set pixel to fg color if bit set
XOR_Done:
        INC   DI                    ;Update pointer
        LOOP XOR_Bit_Loop           ;If not all 16 bits done do next bit
        XOR   BX,8000h              ;Toggle high bit of BX to check if
        JS    Set_XOR_FG            ; both words have been done

        ADD   DI,BX                 ;Point to next scanline
        DEC   DL                    ;Check if all scanlines done
        JG    Set_XOR_FG            ;Go do next scanline if not done

        ; Set 'last cursor' to save area (this is needed for first
        ; call to Move_Cursor procedure, since first thing done in there
        ; is restore area under 'last cursor' position)

        MOV   AX,CS:Save_Area_y     ;Fetch save area y
        MOV   CS:Last_Cursor_y,AX ;Set last cursor y
        MOV   CS:Last_Cursor_x,CUR_OFFSET  ;Set last cursor x

        ; Clean up and return

        POP   DS              ;Restore segment registers
        POP   ES
        POP   DI
        POP   SI

        MOV   SP,BP                 ;Restore stack
        POP   BP
        RET
_Set_Cursor     ENDP
```

```
;************************************************************************
;*                                                                      *
;* _Move_Cursor(Curs_X, Curs_Y)                                         *
;*   This procedure is used to move the cursor from one                 *
;*   location to another. The cursor move is performed using the        *
;*   following steps:                                                   *
;*        1 - Check if new cursor is outside 'cursor block'             *
;*        2 - If outside 'cursor block' restore area under              *
;*            previous block.                                           *
;*            Save area under new block.                                *
;*        3 - Copy saved are into cursor build area (both save and      *
;*            build areas are normally off-screen).                     *
;*        4 - Combine AND and XOR masks with build area.                *
;*        5 - Copy build area to where new cursor should be (this       *
;*            in most cases overwrites the old cursor).                 *
;*   The 'build area' is a rectangle twice the size of the cursor.      *
;*   It is used to eliminate flicker for small movement of the          *
;*   cursor, since cursor may not need to be erased if it moves         *
;*   only by a few pixels.                                              *
;*                                                                      *
;* Entry:                                                               *
;*   Curs_X - Position of the new cursor                                *
;*   Curs_Y                                                             *
;*                                                                      *
;************************************************************************

Arg_Curs_X      EQU  WORD PTR [BP+4] ;Formal parameters
Arg_Curs_Y      EQU  WORD PTR [BP+6]

Curs_X          EQU  WORD PTR [BP-2]
Curs_Y          EQU  WORD PTR [BP-4]

_Move_Cursor    PROC NEAR
     PUSH BP               ;Standard high-level entry
     MOV  BP,SP
     SUB  SP,4

     PUSH SI               ;Save registers
     PUSH DI
     PUSH ES
     PUSH DS

     ; Check if new area needs to be saved

     MOV  AX,Arg_Curs_x       ;Fetch new x
     AND  AX,NOT(CUR_WIDTH-1) ;Round to nearest buffer block
     MOV  BX,Arg_Curs_y       ;Fetch new y
     AND  BX,NOT(CUR_HEIGHT-1);Round to nearest buffer block

     CMP  AX,CS:Last_Cursor_x ;Check if x moved into next block
     JNE  Cursor_New_Block
     CMP  BX,CS:Last_Cursor_y ;Check if y moved into next block
     JNE  Cursor_New_Block
     JMP  Build_Cursor

     ; For new block call to remove old cursor, then use _BitBlt
     ; to save block under next cursor location into the save area

Cursor_New_Block:
     CALL _Remove_Cursor      ;Restore last location
     MOV  AX,Arg_Curs_x       ;Fetch new x
     AND  AX,NOT(CUR_WIDTH-1) ;Round to nearest buffer block
     MOV  CS:Last_Cursor_x,AX ;Save as 'last x'
     MOV  AX,Arg_Curs_y       ;Fetch new y
     AND  AX,NOT(CUR_HEIGHT-1);Round to nearest buffer block
     MOV  CS:Last_Cursor_y,AX ;Save as 'last y'

     MOV  AX,2*CUR_HEIGHT     ;Push width and height
     PUSH AX
     MOV  AX,2*CUR_WIDTH
     PUSH AX
```

```
        PUSH CS:Save_Area_y       ;Push x and y of destination
        MOV  AX,CUR_OFFSET
        PUSH AX
        PUSH CS:Last_Cursor_y     ;Push x and y of source
        PUSH CS:Last_Cursor_x
        CALL _BitBlt
        ADD  SP,12

        ; Use _BitBlt to copy save area into build area

Build_Cursor:
        MOV  AX,2*CUR_HEIGHT      ;Push width and height
        PUSH AX
        MOV  AX,2*CUR_WIDTH
        PUSH AX
        PUSH CS:Save_Area_y       ;Push x and y of destination
        MOV  AX,MIX_OFFSET
        PUSH AX
        PUSH CS:Save_Area_y       ;Push x and y of source
        MOV  AX,CUR_OFFSET
        PUSH AX
        CALL _BitBlt
        ADD  SP,12

        ; Mix AND & XOR masks into build area (this will work only if all of
        ; the save area is in the same segment!!!)

        MOV  CX,Arg_Curs_x        ;Fetch x
        AND  CX,CUR_WIDTH-1       ;Keep 'odd' bits
        ADD  CX,MIX_OFFSET        ;Add 'base x' of save area
        MOV  AX,Arg_Curs_y        ;Fetch y
        AND  AX,CUR_HEIGHT-1      ;Keep 'odd' bits
        ADD  AX,CS:Save_Area_y    ;Add 'base y' of save area
        MUL  CS:Video_Pitch       ;   multiply y by width in bytes
        ADD  AX,CX                ;   add x coordinate to compute offset
        ADC  DX,0                 ;   add overflow to upper 16 bits

        MOV  DI,AX                ;Save offset
        MOV  AL,DL                ;Select page
        CALL Select_Page
        MOV  ES,CS:Graf_Seg       ;Set both segments to video buffer
        MOV  DS,CS:Graf_Seg

        MOV  DL,CUR_HEIGHT        ;Initialize raster counter
        MOV  SI,CS:Save_Offset    ;Get pointer to AND & XOR masks
        MOV  BX,CS:Video_Pitch    ;Compute scan-to-scan increment
        SUB  BX,CUR_WIDTH

Mix_Lines:
        MOV  CX,CUR_WIDTH         ;Fetch cursor width
Mix_Bytes:
        LODSB                     ;Fetch next byte of AND mask
        MOV  AH,[DI]              ;Fetch next byte of destination
        AND  AL,AH                ;Combine mask with destination
        MOV  AH,[SI+CUR_WIDTH-1]  ;Fetch next byte of XOR mask
        XOR  AL,AH                ;Combine with previous result
        STOSB                     ;Place result into destination
        LOOP Mix_Bytes

        ADD  DI,BX                ;Point to next scanline
        ADD  SI,BX                ;Point to next scanline
        DEC  DL                   ;Check if all scanlines done
        JG   Mix_Lines            ;Go do next scanline if not done

        ; Use _BitBlt procedure to copy build area to screen (and erase old
        ; cursor with the new cursor block).

        MOV  AX,2*CUR_HEIGHT      ;Push width and height
        PUSH AX
        MOV  AX,2*CUR_WIDTH
        PUSH AX
```

```
     PUSH CS:Last_Cursor_y    ;Push x and y of destination
     PUSH CS:Last_Cursor_x
     PUSH CS:Save_Area_y      ;Push x and y of source
     MOV  AX,MIX_OFFSET
     PUSH AX
     CALL _BitBlt
     ADD  SP,12

     ; Clean up and return

     POP  DS             ;Restore segment registers
     POP  ES
     POP  DI
     POP  SI

     MOV  SP,BP               ;Restore stack
     POP  BP
     RET
_Move_Cursor   ENDP

;*************************************************************************
;*                                                                       *
;* _Remove_Cursor                                                        *
;*   This procedure is used to remove the cursor from the screen         *
;*   and to restore the screen to its original appearance               *
;*                                                                       *
;*************************************************************************

_Remove_Cursor PROC NEAR
     PUSH BP             ;Standard high-level entry
     MOV  BP,SP

     PUSH SI             ;Save registers
     PUSH DI
     PUSH ES
     PUSH DS

     ; Use _BitBlt to restore area under the last cursor location

     MOV  AX,2*CUR_HEIGHT     ;Push width and height
     PUSH AX
     MOV  AX,2*CUR_WIDTH
     PUSH AX
     PUSH CS:Last_Cursor_y    ;Push last position of cursor
     PUSH CS:Last_Cursor_x
     PUSH CS:Save_Area_y      ;Push x and y of destination
     MOV  AX,CUR_OFFSET
     PUSH AX
     CALL _BitBlt
     ADD  SP,12

     ; Clean up and return

     POP  DS                  ;Restore segment registers
     POP  ES
     POP  DI
     POP  SI

     MOV  SP,BP               ;Restore stack
     POP  BP
     RET
_Remove_Cursor ENDP

_TEXT     ENDS
          END
```

# Load DACs

This module is used to control the color mapping between data in display memory and colors seen on the screen. For 256-color modes this is best done by changing the DAC registers. In most cases changing DAC registers is fast enough so that 'snow' on the screen is not noticeable. However, for applications which require the frequent changing of DAC registers, register updates should be synchronized with vertical retrace. IBM recommends that interrupts be disabled between register selection and register read/writes in order to minimize the time required for a register update.

BIOS function 10h, subfunction 10h or 12h can also be used to modify DAC registers.

**Listing 7-9. File: 256COL\DAC.ASM**

```
;*********************************************************************
;*                                                                   *
;* File:          DAC.ASM - Load DAC registers, Read DAC registers   *
;* Routines:      _Write_DAC,_Read_DAC                               *
;* Arguments:     Start, Count, ArrayPtr                             *
;*                                                                   *
;*********************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR

        PUBLIC  _Read_DAC
        PUBLIC  _Write_DAC

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

;*********************************************************************
;*                                                                   *
;* _Read_DAC(Start, Count, ArrayPtr)                                 *
;*      Read 'Count' DAC registers as RGB triplets into array        *
;*      pointed to by 'ArrayPtr', starting with 'Start' register.    *
;*                                                                   *
;*********************************************************************

Arg_Start       EQU     WORD PTR [BP+4]
Arg_Count       EQU     WORD PTR [BP+6]
Arg_ArrayPtr    EQU     DWORD PTR [BP+8]

_Read_DAC       PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Read the DAC registers

        LES     DI,Arg_ArrayPtr         ;First triplet
        MOV     AX,Arg_Start            ;First register to read
        MOV     CX,Arg_Count            ;Number of registers to read
        MOV     DX,3C7h                 ;Select first DAC register
        OUT     DX,AL
        INC     DX                      ;Set DAC data register
        INC     DX
```

```
DAC_In_Loop:
        IN      AL,DX                   ;Read red
        STOSB                           ;Save into buffer
        IN      AL,DX                   ;Read green
        STOSB                           ;Save green
        IN      AL,DX                   ;Read blue
        STOSB                           ;Save blue
        LOOP    DAC_In_Loop

        ; Cleanup and return

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Read_DAC       ENDP

;*********************************************************************
;*                                                                   *
;* _Write_DAC(Start, Count, ArrayPtr)                                *
;*      Load DAC registers with 'Count' RGB tripplets from array     *
;*      pointed to by 'ArrayPtr', starting with 'Start' register.    *
;*                                                                   *
;*********************************************************************

Arg_Start       EQU     WORD PTR [BP+4]
Arg_Count       EQU     WORD PTR [BP+6]
Arg_ArrayPtr    EQU     DWORD PTR [BP+8]

_Write_DAC      PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Write the registers

        LDS     SI,Arg_ArrayPtr         ;First triplet
        MOV     AX,Arg_Start            ;First register to read
        MOV     CX,Arg_Count            ;Number of registers to read
        MOV     DX,3C8h                 ;Select first DAC register
        OUT     DX,AL
        INC     DX                      ;Set DAC data register
DAC_Out_Loop:
        LODSB                           ;Fetch red
        OUT     DX,AL                   ;Write red
        LODSB                           ;Fetch green
        OUT     DX,AL                   ;Write green
        LODSB                           ;Fetch blue
        OUT     DX,AL                   ;Write blue
        LOOP    DAC_Out_Loop

        ; Cleanup and return

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Write_DAC      ENDP

_TEXT   ENDS
        END
```

# Read Raster Line

_Read_Video and _Write_Video, shown in the next example, can used to save the contents of the display memory and to display stored images. For 256-color modes these procedures are straightforward. The process is analogous to BITBLT except that no checks for page boundaries are needed for system memory, no intermediate buffer or dual paging is needed, and checks for source and destination overlap are not needed. In each procedure, the address of the starting point is computed first, and then the data is copied one scan line at a time.

**Listing 7-10. File: 256COL\READ.ASM**

```
;***********************************************************************
;*                                                                     *
;* File:        READ.ASM - 8bit packed read block into system memory   *
;* Description: Read specified block from video memory and copy each   *
;*              pixel into one byte starting at 'Dest_Ptr'. Next row   *
;*              of the block is copied to 'Dest_Ptr+Dest+Pitch', and   *
;*              so on until the full block is read.                    *
;* Routine:     _Read_Video                                            *
;* Arguments:   x, y, dx, dy, Dest_Pitch, Dest_Ptr                     *
;*                                                                     *
;***********************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Video_Pitch:WORD
        EXTRN   Select_Page:NEAR

        PUBLIC  _Read_Video

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU     WORD PTR [BP+4]
Arg_y           EQU     WORD PTR [BP+6]
Arg_dx          EQU     WORD PTR [BP+8]
Arg_dy          EQU     WORD PTR [BP+10]
Arg_Dest_Pitch  EQU     WORD PTR [BP+12]
Arg_Dest_Ptr    EQU     DWORD PTR [BP+14]

PageNo          EQU     BYTE PTR [BP-2]

_Read_Video     PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer
        SUB     SP,2                    ;Allocate space for local variables

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Compute address of first pixel

        MOV     AX,Arg_y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;   multiply by width in bytes
        ADD     AX,Arg_x                ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits
        MOV     SI,AX                   ;Save offset
        MOV     AL,DL                   ;Select page were first pixel is
        MOV     PageNo,AL               ;Save page number
        CALL    Select_Page
```

```
        MOV     DS,CS:Graf_Seg
        MOV     BX,CS:Video_Pitch       ;Compute line-to-line increment
        SUB     BX,Arg_dx

        LES     DI,Arg_Dest_Ptr         ;Fetch pointer to destination
        MOV     DX,Arg_Dest_Pitch       ;Compute line increment for dest.
        SUB     DX,Arg_dx

        ; Loop over raster lines to copy data

Scan_Loop:
        MOV     CX,Arg_dx               ;Fetch byte count

        ; Copy from initial page if page boundary may be crossed
        MOV     AX,CX                   ;Check if within page
        ADD     AX,SI
        JNC     Scan_In_Page
        SUB     CX,AX                   ;Number of bytes to do in this page
        SHR     CX,1                    ;Adjust for move of words
        REP     MOVSW                   ;Copy data from initial page
        ADC     CX,CX
        REP     MOVSB
        MOV     CX,AX                   ;Number of bytes to do in next page
        XCHG    AL,PageNo               ;Fetch page number, and preserve AL
        INC     AL                      ;Adjust page number
        CALL    Select_Page             ;Select next page
        XCHG    AL,PageNo               ;Save updated page no., restore AL
        JCXZ    Scan_Done
        ; Copy from next (or only) page
Scan_In_Page:
        SHR     CX,1                    ; Adjust for move of words
        REP     MOVSW                   ; Write all words of data
        ADC     CX,CX                   ; Write the last odd byte of data
        REP     MOVSB
Scan_Done:

        ADD     DI,DX                   ; Compute ptr to byte in next raster
        ADD     SI,BX
        JC      Fix_Page
        DEC     Arg_dy                  ; check if more rasters to do
        JG      Scan_Loop
        JMP     SHORT End_Read

Fix_Page:
        XCHG    AL,PageNo               ; Fetch page number, and preserve AL
        INC     AL                      ; Update page number
        CALL    Select_Page             ; Compute and select new page number
        XCHG    AL,PageNo               ; Save updated page no., restore AL
        DEC     Arg_dy                  ; check if more rasters to do
        JG      Scan_Loop

        ; Cleanup and return

End_Read:
        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Read_Video     ENDP

_TEXT   ENDS
        END
```

# Write Raster Line

This is a companion module to Read Raster Line.

**Listing 7-11. File: 256COL\WRITE.ASM**

```
;*********************************************************************
;*                                                                   *
;* File:        WRITE.ASM - 8bit packed write block from system memory  *
;* Description: Write specified block into video memory and copy each   *
;*              pixel starting at 'Src_Ptr'.  Next row                *
;*              of the block is copied from 'Src_Ptr+Src_Pitch', and  *
;*              so on until the full block is written.                *
;*              Assues one byte per pixel in source data.            *
;* Routine:     _Write_Video                                          *
;* Arguments:   x, y, dx, dy, Src_Pitch, Src_Ptr                      *
;*                                                                   *
;*********************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Video_Pitch:WORD
        EXTRN   Select_Page:NEAR

        PUBLIC  _Write_Video

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU     WORD PTR [BP+4]
Arg_y           EQU     WORD PTR [BP+6]
Arg_dx          EQU     WORD PTR [BP+8]
Arg_dy          EQU     WORD PTR [BP+10]
Arg_Src_Pitch   EQU     WORD PTR [BP+12]
Arg_Src_Ptr     EQU     DWORD PTR [BP+14]

PageNo          EQU     BYTE PTR [BP-2]

_Write_Video    PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer
        SUB     SP,2                    ;Allocate space for local variables

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Compute address of first pixel

        MOV     AX,Arg_y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;   multiply by width in bytes
        ADD     AX,Arg_x                ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits
        MOV     DI,AX                   ;Save offset
        MOV     AL,DL                   ;Select page were first pixel is
        MOV     PageNo,AL               ;Save page number
        CALL    Select_Page
        MOV     ES,CS:Graf_Seg
        MOV     DX,CS:Video_Pitch       ;Compute line-to-line increment
        SUB     DX,Arg_dx

        LDS     SI,Arg_Src_Ptr          ;Fetch pointer to source
        MOV     BX,Arg_Src_Pitch        ;Compute line increment for dest.
        SUB     BX,Arg_dx

        ; Loop over raster lines to copy data
```

```
Scan_Loop:
        MOV     CX,Arg_dx               ;Fetch byte count

        ; Copy from initial page if page boundary may be crossed
        MOV     AX,CX                   ;Check if within page
        ADD     AX,DI
        JNC     Scan_In_Page
        SUB     CX,AX                   ;Number of bytes to do in this page
        SHR     CX,1                    ;Adjust for move of words
        REP     MOVSW                   ;Copy data from initial page
        ADC     CX,CX
        REP     MOVSB
        MOV     CX,AX                   ;Number of bytes to do in next page
        XCHG    AL,PageNo               ;Fetch page number, and preserve AL
        INC     AL                      ;Adjust page number
        CALL    Select_Page             ;Select next page
        XCHG    AL,PageNo               ;Save updated page no., restore AL
        JCXZ    Scan_Done
        ; Copy from next (or only) page
Scan_In_Page:
        SHR     CX,1                    ; Adjust for move of words
        REP     MOVSW                   ; Write all words of data
        ADC     CX,CX                   ; Write the last odd byte of data
        REP     MOVSB
Scan_Done:

        ADD     SI,BX                   ; Compute ptr to byte in next raster
        ADD     DI,DX
        JC      Fix_Page
        DEC     Arg_dy                  ; check if morerasters to do
        JG      Scan_Loop
        JMP     SHORT End_Write

Fix_Page:
        XCHG    AL,PageNo               ; Fetch page number, and preserve AL
        INC     AL                      ; Update page number
        CALL    Select_Page             ; Compute and select new page number
        XCHG    AL,PageNo               ; Save updated page no., restore AL
        DEC     Arg_dy                  ; check if more rasters to do
        JG      Scan_Loop

        ; Cleanup and return

End_Write:
        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Write_Video    ENDP

_TEXT   ENDS
        END
```

# 8

# Programming Examples 16-Color Graphics

# Introduction

High resolution 16-color graphics modes are useful for applications such as CAD (Computer-Aided Design) where high resolution is more important than number of simultaneous colors. Drawing algorithms for this memory organization tend to be more complex than those for 256-color modes, since each byte of memory contains several pixels. Fill operations and byte-aligned transfers are faster, but incremental algorithms tend to be slower. Procedures that can process several pixels within a byte simultaneously perform better than those that must process individual pixels, which can be excruciatingly slow.

Extended 16-color planar graphics modes are identical in structure to the standard VGA 16-color modes (modes D,E,10 and 12). Two resolutions are common for this memory organization: 800x600 and 1024x768. Resolutions as high as 800x600 can be supported with just 256K of display memory and no memory paging.

For 1024x768 resolution, which requires 512K of display memory, some SuperVGAs (such as those based on the Tseng Labs ET3000) increase the size of the host window into display memory from 64K to 128K, eliminating the need for display memory paging. It is still necessary, however, to detect the 64K segment boundary that lies in the middle of that larger host window. Segment boundary detection is essentially the same as page boundary detection, so most drawing algorithms are not simplified by this method. Boards that do not use a larger host memory window must include a memory paging scheme.

Designed to illustrate basic techniques for graphics programming in 16-color planar pixel modes, the programming examples in this chapter can be used on any VGA board that supports a resolution of 800x600 (no paging) or 1024x768 (page boundary at end of raster) in 16 colors. It should be noted that the programming examples in this chapter will not work properly in cases where page boundaries fall in the middle of a scan line; none of the 16-color modes used on SuperVGAs today have such cases.

These examples show how to draw basic graphics primitives such as pixels, lines and rectangles, and how to perform BITBLT transfers. Also included are routines to draw and erase a software graphics cursor, and routines to load the color palette.

Examples are written in assembly language, and assume that input parameters will be placed on the stack before the routine is called, conforming to the convention for C-callable subroutines.

It is assumed that the VGA will already be initialized to the desired graphics mode before the drawing routine examples are executed.

Some SuperVGA boards include extended 4-color or 16-color display modes that use packed pixels. Examples in this chapter do not apply to these modes; they are described in the appropriate board-dependent chapter.

# Display Memory Organization

Figure 8-1 shows the organization of display memory for these modes. Each pixel occupies one bit position in each plane. To convert from a pixel position, in X and Y coordinates, to a bit location in display memory, use the following equations:

| | |
|---|---|
| Page | = (Video_Pitch x Y + X/8) / 10000h |
| Segment | = A000h |
| Offset | = (Video_Pitch x Y + X/8) mod 10000h |
| Bit position | = X mod 8 |

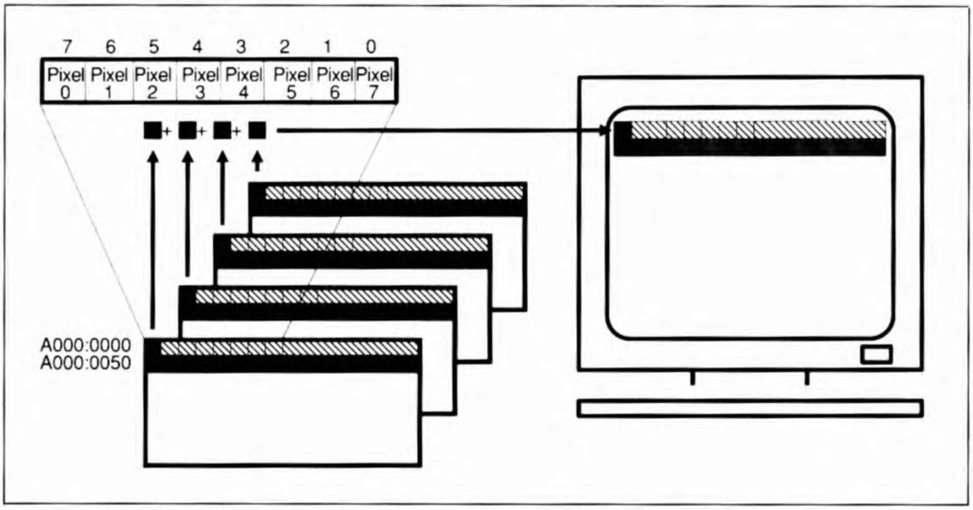


**Figure 8-1. Display memory organization—16-color graphics**

Pixel data is serialized for display most significant bit first, which means that the most significant bit of each byte in display memory represents the leftmost pixel. Each byte contains eight pixels.

# Drawing Routines

## Write Pixel

Write mode 2 of the Graphics Controller is used as a convenient method of writing to all four color planes simultaneously. This mode permits a four-bit color value, formatted as a packed pixel, to be written directly into the four color planes.

_Write_Pixel is a simple example that shows how to access a pixel with screen coordinates x,y. The x,y coordinate is used to compute page and offset using the 16-bit multiply instruction MUL followed by a 32-bit add of the x coordinate divided by 8 (done with three SHR instructions). At the completion of these two operations, register DX contains the page number and register AX contains the offset. Page selection is performed using the board-dependent procedure _Select_Page. The pixel can then be accessed using the offset in AX and the mode-dependent segment variable Graf_Seg. Graphics Controller register index 8 is used to define a mask to enable a single pixel for writing. A pixel mask is computed using the following formula:

Mask = 80h SHR (X AND 7)

To perform masking, the VGA processor read latches are loaded by a display memory read operation. The new pixel value is then written to display memory. Pixel color is defined by the Set/Reset register, which is set and enabled by the procedure _Select_Color. Instead of using Set/Reset, write mode 2 could be used (enabled via Graphics Controller register 5); Set/Reset is more efficient, however, for most drawing algorithms.

**Listing 8-1. File: 16COL\WPIXEL.ASM**

```
;*********************************************************************
;*                                                                   *
;* File:        WPIXEL.ASM - 4 Bit Planar Pixel Write                *
;* Routine:     _Write_Pixel                                         *
;* Arguments:   X, Y, Color                                          *
;*                                                                   *
;* Routine:     Select_Color                                         *
;* Arguments:   AL = Color                                           *
;*                                                                   *
;*********************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pitch:WORD

        PUBLIC  _Write_Pixel
        PUBLIC  Select_Color

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU     WORD PTR [BP+4]
Arg_y           EQU     WORD PTR [BP+6]
Arg_Color       EQU     BYTE PTR [BP+8]

_Write_Pixel    PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Convert x,y pixel addres to Page and Offset
        MOV     AX,Arg_y                ;Fetch y coordinate
```

```
        MUL     CS:Video_Pitch                  multiply by raster width
        MOV     CX,Arg_x                        add x coordinate/8
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     ES,CS:Graf_Seg          ;Put address in ES:DI
        MOV     DI,AX
        MOV     AL,DL                   ;Select proper page
        CALL    Select_Page

        ; Set Graphics Controller for proper color

        MOV     AL,Arg_Color            ;Fetch color to use
        CALL    Select_Color            ;Select color

        ; Set write mask

        MOV     CX,Arg_x                ;Compute X AND 7 to find mask rotation
        AND     CX,7                    ;Mask rotation is now in CL
        MOV     AL,80h                  ;Shift bit to find mask
        SHR     AL,CL                   ;Mask is now in AL
        MOV     DX,GRAPHICS_CTRL_PORT   ;Fetch graphics controller port
        MOV     AH,AL                   ;Put mask in AH
        MOV     AL,BIT_MASK_REG         ;Select bit mask register
        OUT     DX,AX                   ;Set bit mask

        ; Set pixel

        MOV     AH,ES:[DI]              ;Latch previous value
        MOV     ES:[DI],AL              ;Write color (using set/reset)

        ; Cleanup and exit

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Write_Pixel    ENDP

;************************************************************************
;*                                                                      *
;* Routine:     Select_Color                                            *
;*              Utility routine used by all drawing routines to select *
;*              specified color.  It is assumed that all planes are    *
;*              enabled for write, and that 'processor write' mode is  *
;*              selected.  Routine enables set/reset mechanism of VGA. *
;* Arguments:   AL = Color                                              *
;* Returns:     DX = Points to mask select data register               
;*                                                                      *
;************************************************************************

Select_Color    PROC NEAR
        PUSH    AX
        MOV     DX,GRAPHICS_CTRL_PORT   ;Use color for set/reset value
        MOV     AH,AL
        MOV     AL,SET_RESET_REG
        OUT     DX,AX
        MOV     DX,GRAPHICS_CTRL_PORT   ;Enable set/reset
        MOV     AL,SR_ENABLE_REG
        MOV     AH,0Fh
        OUT     DX,AX
        MOV     AL,BIT_MASK_REG         ;Select bit mask register
        OUT     DX,AL
        INC     DX
        POP     AX
```

```
        RET
Select_Color    ENDP

_TEXT   ENDS
        END
```

# Read Pixel

_Read_Pixel is a companion to the _Write_Pixel procedure. Computation of the pixel address and mask is the same as for pixel write. To read the pixel using a planar memory organization, each of the four bits in a pixel must be read from a separate plane using a separate read instruction. Before each read, the proper color plane must be enabled. Each bit must also be properly masked and rotated. The result is a very complex routine to perform a very simple function.

**Listing 8-2. File: 16COL\RPIXEL.ASM**

```
;**********************************************************************
;*                                                                    *
;* File:        RPIXEL.ASM - 4 Bit Planar Pixel Read                  *
;* Routine:     _Read_Pixel                                           *
;* Arguments:   X, Y                                                  *
;* Returns:     Color in AX                                           *
;*                                                                    *
;**********************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pitch:WORD

        PUBLIC  _Read_Pixel

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU     WORD PTR [BP+4]
Arg_y           EQU     WORD PTR [BP+6]

_Read_Pixel     PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Convert x,y pixel addres to Page and Offset

        MOV     AX,Arg_y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;       multiply by raster width
        MOV     CX,Arg_x                ;       add x coordinate/8
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     ES,CS:Graf_Seg          ;Put address in ES:DI
        MOV     DI,AX
        MOV     AL,DL                   ;Select proper page
        CALL    Select_Page

        ; Setup to read the value at the computed address
```

```
        MOV     DX,GRAPHICS_CTRL_PORT   ;Select Read Plane register
        MOV     AL,READ_PLANE_REG
        OUT     DX,AL
        INC     DX                      ;Point DX to data register

        MOV     AL,3                    ;Plane number
        MOV     CX,Arg_x                ;Compute X AND 7 to find mask rotation
        AND     CX,7                    ;Mask rotation is now in CL
        MOV     BL,80h                  ;Shift bit to find mask
        SHR     BL,CL                   ;Mask is now in BL
        XOR     BH,BH                   ;Initialize return value to zero

Plane_Loop:

        OUT     DX,AL                   ;Select plane n for reading (from AL)

        ; Read byte, mask correct bit and add it into the return value

        SHL     BH,1                    ;Shift return value up
        MOV     AH,ES:[DI]              ;Get byte of video memory
        AND     AH,BL                   ;Mask out unwanted bits
        JZ      RP_Not_Set              ;Jump if bit not set
        OR      BH,1                    ;Set bit in return value
RP_Not_Set:
        DEC     AL                      ;Decrement plane number
        JGE     Plane_Loop              ;Do another plane if there are more
        MOV     AL,BH                   ;Put return value in AL
        XOR     AH,AH                   ;Clear AH

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Read_Pixel     ENDP

_TEXT   ENDS
        END
```

## Draw Solid Line

_Line is used to demonstrate techniques used in incremental algorithms. An initial page and offset is computed from the starting x,y coordinate of the line. The line is then classified according to its slope (the relative size of DX and DY), and whether x and y are increasing or decreasing. Each line will fall into one of eight different classes, with different sections of code applying to each class.

Although some code sections could be combined to reduce total code size, the code is left in eight distinct sections to make it easier to add patterns and 'last pixel don't draw' checks. Each of the eight sections is divided into two parts: incremental drawing and page updating. For example, lines with positive DX and DY and DX greater than DY use the incremental drawing code between the labels XP_YP_Next and XP_YP_Update_Seg.

This code is a standard adaptation of Bresenham's line drawing algorithm, but with an added JC instruction for page boundary detection after y is updated (ADD DI,Pitch).

Two additional code sections are added: one for vertical lines (DX = 0) and one of horizontal lines (DY = 0). For horizontal lines this provides significant performance improvement, since eight pixels are drawn for each write to display memory.

### Listing 8-3. File: 16COL\LINE.ASM

```
;************************************************************************
;*                                                                      *
;* File:        LINE.ASM - 4 Bit Planar Solid Line                     *
;* Routine:     _Line                                                   *
;* Arguments:   X0, Y0, X1, Y1, Color                                   *
;*                                                                      *
;************************************************************************

        INCLUDE VGA.INC

        EXTRN   Select_Color:NEAR

        EXTRN   Video_Pitch:WORD
        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR

        PUBLIC  _Line

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_X0          EQU     WORD PTR [BP+4] ;Formal parameters
Arg_Y0          EQU     WORD PTR [BP+6]
Arg_X1          EQU     WORD PTR [BP+8]
Arg_Y1          EQU     WORD PTR [BP+10]
Arg_Color       EQU     BYTE PTR [BP+12]

D1              EQU     WORD PTR [BP-2] ;Local variables
D2              EQU     WORD PTR [BP-4]
Pitch           EQU     WORD PTR [BP-6]
Delta_X         EQU     WORD PTR [BP-8]
First_Mask      EQU     BYTE PTR [BP-9]
PageNo          EQU     BYTE PTR [BP-10]

_Line           PROC NEAR
        PUSH    BP                      ;Standard C entry point
        MOV     BP,SP
        SUB     SP,10                   ;Declare local variables

        PUSH    DI                      ;Preserve segment registers
        PUSH    SI
        PUSH    DS
        PUSH    ES

;-----------------------------------------------------------------
; Compute address of first pixel, select color
;-----------------------------------------------------------------

        MOV     AX,Arg_Y0               ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;       multiply by raster width
        MOV     CX,Arg_X0               ;       add x coordinate/8
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        PUSH    AX                      ;Save offset within page
        MOV     DS,CS:Graf_Seg          ;Fetch segment
        MOV     ES,CS:Graf_Seg          ;Fetch segment
        MOV     AL,DL                   ;Select proper page
        MOV     PageNo,AL               ;Save page number for later
        CALL    Select_Page
```

```
        MOV     CX,Arg_X0               ;Fetch x0
        AND     CL,7                    ;Get bit position within first byte
        MOV     BL,80h                  ;Assume first bit
        SHR     BL,CL                   ;Rotate mask bit into place
        MOV     First_Mask,BL           ;Save mask

        ; Load set/reset registers with current color, select bit mask reg

        MOV     AL,Arg_Color
        CALL    Select_Color            ;Also sets DX to gr. ctrl. data register

;-------------------------------------------------------------------
; Compute dx and dy and determine which coordinate is major
;-------------------------------------------------------------------

        MOV     AX,CS:Video_Pitch       ;Set raster increment
        MOV     Pitch,AX
        MOV     SI,Arg_X1               ;Compute dx (X1-X0) in SI
        SUB     SI,Arg_X0
        MOV     Delta_X,SI              ;Save in local variable
        JGE     DX_Pos                  ;If dx is negative, make it positive
        NEG     SI
DX_Pos:
        MOV     DI,Arg_Y1               ;Compute dy (Y1-Y0) in DI
        SUB     DI,Arg_Y0
        JGE     DY_Pos                  ;If dy is negative, make it positive
        NEG     Pitch                   ;Also, invert the pitch
        NEG     DI
DY_Pos:

        ; Figure out which coordinate is the major one

        OR      SI,SI                   ;Check for vertical line
        JE      Vertical
        OR      DI,DI                   ;Check for horizontal line
        JE      Horizontal
        CMP     SI,DI                   ;Check that dx > dy
        JL      Y_Major_Jump
        JMP     X_Major
Y_Major_Jump:
        JMP     Y_Major

;-------------------------------------------------------------------
; Vertical Line
;-------------------------------------------------------------------

Vertical:
        MOV     CX,DI                   ;Set up counter
        INC     CX                      ;Number of pixels is one greater
        MOV     AL,First_Mask           ;Fetch mask
        OUT     DX,AL                   ;Set mask

        POP     DI                      ;Fetch offset
        MOV     BX,Pitch                ;Fetch pitch
        OR      BX,BX                   ;Check for y decreasing
        JNS     Vert_Loop

        ; Y1 < Y0, but we want to draw down only, so compute address of
        ; (X1,Y1) and start from there

        NEG     BX
        XCHG    SI,CX                   ;Preserve counter

        MOV     AX,Arg_Y1               ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;       multiply by raster width
        MOV     CX,Arg_X1               ;       add x coordinate/8
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
```

```
        MOV     DI,AX                   ;Save offset within page
        MOV     ES,CS:Graf_Seg          ;Fetch segment
        MOV     AL,DL                   ;Select proper page
        MOV     PageNo,AL               ;Save page number for later
        CALL    Select_Page

        XCHG    SI,CX                   ;Restore counter

Vert_Loop:
        NOT     BYTE PTR [DI]           ;Latch data, then set to new value
        ADD     DI,BX                   ;Update offset
        JC      Vert_Update_Seg         ;Update segment if carry
        LOOP    Vert_Loop
        JMP     End_Line

Vert_Update_Seg:
        PUSH    AX
        INC     PageNo                  ;Advance page number
        MOV     AL,PageNo               ;Select next page
        CALL    Select_Page
        POP     AX
        LOOP    Vert_Loop
        JMP     End_Line

;-----------------------------------------------------------------
; Horizontal line
;-----------------------------------------------------------------

Horizontal:
        POP     DI                      ;Fetch offset

        ; Draw pixels from the leading partial byte

        MOV     AX,Arg_X0               ;Fetch x coordinate (assume x1 > x0)
        CMP     Delta_X,0               ;Is x1 > x0?
        JNS     Horz_In_Order

        ; X1 < X0, but we want to draw right only, so compute address of
        ; (X1,Y1) and start from there

        MOV     AX,Arg_Y1               ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;       multiply by raster width
        MOV     CX,Arg_X1               ;       add x coordinate/8
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     DI,AX                   ;Save offset within page
        MOV     ES,CS:Graf_Seg          ;Fetch segment
        MOV     AL,DL                   ;Select proper page
        MOV     PageNo,AL               ;Save page number for later
        CALL    Select_Page

        MOV     DX,GRAPHICS_CTRL_PORT+1 ;Put gr ctrl data register in DX
        MOV     AX,Arg_X1

Horz_In_Order:
        MOV     CX,SI                   ;Set counter of pixels
        INC     CX
        AND     AX,7                    ;Check for partial byte
        JZ      Horz_Full
        MOV     BL,0FFh                 ;Compute the mask
        XCHG    BH,CL                   ;Preserve counter (CL into BH)
        MOV     CL,AL
        SHR     BL,CL
        XCHG    BH,CL                   ;Restore counter
        ADD     CX,AX                   ;Update counter
        SUB     CX,8
        JGE     Mask_Set                ;Modify mask if only one byte
        NEG     CX
```

```
        SHR     BL,CL
        SHL     BL,CL
        XOR     CX,CX                   ;Set bit count to zero
Mask_Set:
        MOV     AL,BL                   ;Fetch mask
        OUT     DX,AL                   ;Set mask register
        MOV     AH,ES:[DI]              ;Latch data
        STOSB                           ;Write new data

        ; Draw the middle complete bytes

Horz_Full:
        MOV     BX,CX                   ;Check if any bytes to set
        CMP     BX,8
        JL      Horz_Trailing
        SHR     CX,1                    ;Compute count
        SHR     CX,1
        SHR     CX,1
        MOV     AL,0FFh
        OUT     DX,AL
        REP     STOSB

        ; Draw the trailing partial byte

Horz_Trailing:
        AND     BL,7
        JZ      Horz_Done
        MOV     AL,0FFh                 ;Compute mask
        MOV     CX,BX
        SHR     AL,CL
        NOT     AL
        OUT     DX,AL                   ;Set the mask
        MOV     AL,ES:[DI]              ;Latch data
        STOSB                           ;Set new data

Horz_Done:
        JMP     End_Line

;-----------------------------------------------------------------
; Diagonal line for x-major
;-----------------------------------------------------------------

        ; Compute constants for x-major

X_Major:
        MOV     CX,SI                   ;Set counter to dx+1
        INC     CX
        SAL     DI,1                    ;D1 = dy*2
        MOV     BX,DI                   ;D  = dy*2-dx
        SUB     BX,SI
        NEG     SI                      ;D2 = dy*2-dx-dx
        ADD     SI,BX
        MOV     D1,DI                   ;Save d1
        MOV     D2,SI                   ;Save d2
        POP     DI                      ;Restore offset of first pixel
        MOV     AL,First_Mask           ;Fetch the initial mask
        XOR     SI,SI                   ;Keep 0 in SI

        ; Jump according to sign of dx and dy

        OR      Pitch,SI                ;Check if dy is positive
        JNS     XM_Y_Pos
        NEG     Pitch                   ;Restore pitch
        OR      Delta_X,SI
        JS      XNYN_Jump
        JMP     XPYN                    ;Go do dy negative dx positive
XNYN_Jump:
        JMP     XNYN                    ;Go do both dy and dx negative
XM_Y_Pos:
        OR      Delta_X,SI              ;Check if dx also positive
        JNS     XPYP                    ;Jump if both dx and dy are positive
```

```
        JMP     XNYP                    ;Jump if dx is negative and dy positive

        ;-------------------------------------------------------------
        ; Draw line where DX > 0 and DY > 0 and x major
        ;-------------------------------------------------------------

XPYP:
XPYP_Next:                              ;Loop over pixels to be set
        OUT     DX,AL                   ;Set next mask
        NOT     BYTE PTR [DI]           ;Latch data, then set to new value
        ROR     AL,1                    ;Update mask
        ADC     DI,SI                   ;and address (if needed)
        OR      BX,BX                   ;If d >= 0 then ...
        JS      XPYP_D_Neg
        ADD     BX,D2                   ;... d = d + d2
        ADD     DI,Pitch                ;Update offset
        JC      XPYP_Update_Seg
        LOOP    XPYP_Next
        JMP     End_Line

XPYP_D_Neg:
        ADD     BX,D1                   ;If d < 0 then d = d + d1
        LOOP    XPYP_Next
        JMP     End_Line

XPYP_Update_Seg:
        PUSH    AX
        INC     PageNo                  ;Select next page
        MOV     AL,PageNo
        CALL    Select_Page
        POP     AX
        LOOP    XPYP_Next
        JMP     End_Line

        ;-------------------------------------------------------------
        ; Draw line where DX < 0 and DY > 0 and X major
        ;-------------------------------------------------------------

XNYP:
XNYP_Next:                              ;Loop over pixels to be set
        OUT     DX,AL                   ;Set next mask
        NOT     BYTE PTR [DI]           ;Latch data, then set to new value
        ROL     AL,1                    ;Update mask
        SBB     DI,SI                   ;and address (if needed)
        OR      BX,BX                   ;If d >= 0 then ...
        JS      XNYP_D_Neg
        ADD     BX,D2                   ;... d = d + d2
        ADD     DI,Pitch                ;Update offset
        JC      XNYP_Update_Seg         ;Update page number if needed
        LOOP    XNYP_Next
        JMP     End_Line

XNYP_D_Neg:
        ADD     BX,D1                   ;If d < 0 then d = d + d1
        LOOP    XNYP_Next
        JMP     End_Line

XNYP_Update_Seg:
        PUSH    AX
        INC     PageNo                  ;Select next page
        MOV     AL,PageNo
        CALL    Select_Page
        POP     AX
        LOOP    XNYP_Next
        JMP     End_Line

        ;-------------------------------------------------------------
        ; Draw line where DX > 0 and DY < 0 and x major
        ;-------------------------------------------------------------

XPYN:
```

```
XPYN_Next:                                  ;Loop over pixels to be set
        OUT     DX,AL                       ;Set next mask
        NOT     BYTE PTR [DI]               ;Latch data, then set to new value
        ROR     AL,1                        ;Update mask
        ADC     DI,SI                       ;and address (if needed)
        OR      BX,BX                       ;If d >= 0 then ...
        JS      XPYN_D_Neg
        ADD     BX,D2                       ;... d = d + d2
        SUB     DI,Pitch                    ;Update offset
        JC      XPYN_Update_Seg             ;Update page number if needed
        LOOP    XPYN_Next
        JMP     End_Line

XPYN_D_Neg:
        ADD     BX,D1                       ;If d < 0 then d = d + d1
        LOOP    XPYN_Next
        JMP     End_Line

XPYN_Update_Seg:
        PUSH    AX
        DEC     PageNo                      ;Select previous page
        MOV     AL,PageNo
        CALL    Select_Page
        POP     AX
        LOOP    XPYN_Next
        JMP     End_Line

        ;-------------------------------------------------------------
        ; Draw line where DX < 0 and DY < 0 and x major
        ;-------------------------------------------------------------

XNYN:
XNYN_Next:                                  ;Loop over pixels to be set
        OUT     DX,AL                       ;Set next mask
        NOT     BYTE PTR [DI]               ;Latch data, then set to new value
        ROL     AL,1                        ;Update mask
        SBB     DI,SI                       ;and address (if needed)
        OR      BX,BX                       ;If d >= 0 then ...
        JS      XNYN_D_Neg
        ADD     BX,D2                       ;... d = d + d2
        SUB     DI,Pitch                    ;Update offset
        JC      XNYN_Update_Seg             ;Update page number if needed
        LOOP    XNYN_Next
        JMP     End_Line

XNYN_D_Neg:
        ADD     BX,D1                       ;If d < 0 then d = d + d1
        LOOP    XNYN_Next
        JMP     End_Line

XNYN_Update_Seg:
        PUSH    AX
        DEC     PageNo                      ;Select previous page
        MOV     AL,PageNo
        CALL    Select_Page
        POP     AX
        LOOP    XNYN_Next
        JMP     End_Line

;-------------------------------------------------------------------
; Diagonal line for y-major
;-------------------------------------------------------------------

        ; Compute constants for dx < dy

Y_Major:
        MOV     CX,DI                       ;Set counter to dy+1
        INC     CX
        SAL     SI,1                        ;D1 = dx * 2
        MOV     BX,SI                       ;D  = dx * 2 - dy
        SUB     BX,DI
```

```
        NEG     DI                      ;D2 = -dy + dx * 2 - dy
        ADD     DI,BX
        MOV     d2,DI                   ;Save d2
        MOV     d1,SI                   ;Save d1
        POP     DI                      ;Restore address of first pixel
        MOV     AL,First_Mask           ;Fetch mask
        XOR     SI,SI                   ;Keep 0 in SI

        ; Jump according to sign of dx and dy

        OR      Pitch,SI                ;Check if dy is positive
        JNS     YM_Y_Pos
        NEG     Pitch
        OR      Delta_X,SI
        JS      NXNY_Jump
        JMP     PXNY                    ;Go do dy negative dx positive
NXNY_Jump:
        JMP     NXNY                    ;Go do both dy and dx negative

YM_Y_Pos:
        OR      Delta_X,SI              ;Check if dx also positive
        JNS     PXPY                    ;Jump if both dx and dy are positive
        JMP     NXPY                    ;Jump if dx is negative and dy positive

        ;-------------------------------------------------------------
        ; Draw line where DX > 0 and DY > 0 and y major
        ;-------------------------------------------------------------

PXPY:
        OUT     DX,AL                   ;Set mask
PXPY_Next:
        NOT     BYTE PTR [DI]           ;Latch data, then set to new value
        ADD     DI,Pitch                ;Update offset
        JC      PXPY_Update_Seg         ;Update page number if needed
PXPY_Updated:
        OR      BX,BX                   ;If d >= 0 then ...
        JS      PXPY_D_Neg

        ADD     BX,D2                   ;... d = d + d2
        ROR     AL,1                    ;Update mask
        OUT     DX,AL                   ;Set mask
        ADC     DI,SI                   ;and address (if needed)
        LOOP    PXPY_Next
        JMP     End_Line

PXPY_D_Neg:
        ADD     BX,D1                   ;If d < 0 then d = d + d1
        LOOP    PXPY_Next
        JMP     End_Line

PXPY_Update_Seg:
        PUSH    AX
        INC     PageNo                  ;Select next page
        MOV     AL,PageNo
        CALL    Select_Page
        POP     AX
        JMP     PXPY_Updated

        ;-------------------------------------------------------------
        ; Draw line where DX < 0 and DY > 0 and y major
        ;-------------------------------------------------------------

NXPY:
        OUT     DX,AL                   ;Set mask
NXPY_Next:
        NOT     BYTE PTR [DI]           ;Latch data, then set to new value
        ADD     DI,Pitch                ;Update offset
        JC      NXPY_Update_Seg         ;Update page number if needed
NXPY_Updated:
        OR      BX,BX                   ;If d >= 0 then
        JS      NXPY_D_Neg
```

```
        ADD     BX,D2                   ;... d = d + d2
        ROL     AL,1                    ;Update mask
        OUT     DX,AL                   ;Set mask
        SBB     DI,SI                   ;and address (if needed)
        LOOP    NXPY_Next
        JMP     End_Line

NXPY_D_Neg:
        ADD     BX,D1                   ;If d < 0 then d = d + d1
        LOOP    NXPY_Next
        JMP     End_Line

NXPY_Update_Seg:
        PUSH    AX
        INC     PageNo                  ;Select next page
        MOV     AL,PageNo
        CALL    Select_Page
        POP     AX
        JMP     NXPY_Updated

        ;-----------------------------------------------------------------
        ; Draw line where DX > 0 and DY < 0 and y major
        ;-----------------------------------------------------------------

PXNY:
        OUT     DX,AL                   ;Set mask
PXNY_Next:
        NOT     BYTE PTR [DI]           ;Latch data, then set to new value
        SUB     DI,Pitch                ;Update offset
        JC      PXNY_Update_Seg         ;Update page number if needed
PXNY_Updated:
        OR      BX,BX                   ;If d >= 0 then
        JS      PXNY_D_Neg

        ADD     BX,D2                   ;... d = d + d2
        ROR     AL,1                    ;Update mask
        OUT     DX,AL                   ;Set mask
        ADC     DI,SI                   ;and address (if needed)
        LOOP    PXNY_Next
        JMP     End_Line

PXNY_D_Neg:
        ADD     BX,D1                   ;If d < 0 then d = d + d1
        LOOP    PXNY_Next
        JMP     End_Line

PXNY_Update_Seg:
        PUSH    AX
        DEC     PageNo                  ;Select previous page
        MOV     AL,PageNo
        CALL    Select_Page
        POP     AX
        JMP     PXNY_Updated

        ;-----------------------------------------------------------------
        ; Draw line where DX < 0 and DY < 0 and y major
        ;-----------------------------------------------------------------

NXNY:
        OUT     DX,AL                   ;Set mask
NXNY_Next:
        NOT     BYTE PTR [DI]           ;Latch data, then set to new value
        SUB     DI,Pitch                ;Update offset
        JC      NXNY_Update_Seg         ;Update page number if needed
NXNY_Updated:
        OR      BX,BX                   ;If d >= 0 then ...
        JS      NXNY_D_Neg

        ADD     BX,D2                   ;... d = d + d2
        ROL     AL,1                    ;Update mask
        OUT     DX,AL                   ;Set mask
```

```
        SBB     DI,SI                   ;and address (if needed)
        LOOP    NXNY_Next
        JMP     End_Line

NXNY_D_Neg:
        ADD     BX,D1                   ;If d < 0 then d = d + d1
        LOOP    NXNY_Next
        JMP     End_Line

NXNY_Update_Seg:
        PUSH    AX
        DEC     PageNo                  ;Select previous page
        MOV     AL,PageNo
        CALL    Select_Page
        POP     AX
        JMP     NXNY_Updated

;-----------------------------------------------------------------
; Clean up and return to caller
;-----------------------------------------------------------------

End_Line:
        POP     ES                      ;Restore segment registers
        POP     DS
        POP     SI
        POP     DI

        MOV     SP,BP                   ;Standard C exit point
        POP     BP
        RET
_Line           ENDP

_TEXT   ENDS
        END
```

## Draw Scan Line

Scan line fill is a key building block in most fill algorithms. In the programming example _Scan_Line, the input coordinates are first ordered so that X0 < X1, and the starting coordinate X0,Y is translated to Page:Offset. Scan line drawing must be performed in three parts, since a scan line can start in the middle of a byte, and can end in a middle of a byte. First, the leading partial byte is drawn if needed, then some number of full bytes are drawn, and finally the trailing partial byte is drawn if needed.

No page boundary checking is performed, since for all commonly found 16-color modes there are never any page boundaries in mid-scanline.

**Listing 8-4. File: 16COL\SCANLINE.ASM**

```
;***********************************************************************
;*                                                                     *
;* File:        SCANLINE.ASM - 4 Bit Planar Scan Line                  *
;* Routine:     _Scan_Line                                             *
;* Arguments:   X0, X1, Y, Color                                       *
;* Returns:     Color in AX                                            *
;*                                                                     *
;***********************************************************************

        INCLUDE VGA.INC
        EXTRN   Select_Color:NEAR
        EXTRN   Video_Pitch:WORD
        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
```

```
        PUBLIC  _Scan_Line

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_X0          EQU     WORD PTR [BP+4] ;Formal parameters
Arg_X1          EQU     WORD PTR [BP+6]
Arg_Y           EQU     WORD PTR [BP+8]
Arg_Color       EQU     BYTE PTR [BP+10]

_Scan_Line      PROC NEAR
        PUSH    BP
        MOV     BP,SP

        PUSH    DI
        PUSH    SI
        PUSH    DS
        PUSH    ES

        MOV     AX,Arg_X0               ;Make sure that x1 >= x0
        MOV     CX,Arg_X1
        CMP     CX,AX
        JGE     In_Order
        MOV     Arg_X0,CX
        MOV     Arg_X1,AX

;-----------------------------------------------------------------
; Compute address of first pixel, load set/reset (color) register
;-----------------------------------------------------------------

        ; Compute page number and select the page

In_Order:
        MOV     AX,Arg_Y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;       multiply by raster width
        MOV     CX,Arg_X0               ;       add x coordinate/8
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     DI,AX                   ;Save offset within page
        MOV     ES,CS:Graf_Seg          ;Fetch segment
        MOV     AL,DL                   ;Select proper page
        CALL    Select_Page

        ; Load set/reset registers with current color, select bit mask reg

        MOV     AL,Arg_Color            ;Fetch color to use
        CALL    Select_Color            ;Also sets DX to gr ctrl data register

        MOV     CX,Arg_X1               ;Compute number of pixels to draw
        SUB     CX,Arg_X0
        INC     CX

;-----------------------------------------------------------------
; Draw the scanline
;-----------------------------------------------------------------

        ; Draw pixels from the leading partial byte

        MOV     AX,Arg_X0               ;Fetch x coordinate
        AND     AX,7                    ;Check for partial byte
        JZ      Full
        MOV     BL,0FFh                 ;Compute the mask
        XCHG    BH,CL                   ;Preserve counter (CL into BH)
        MOV     CL,AL
        SHR     BL,CL
        XCHG    BH,CL                   ;Restore counter
        ADD     CX,AX                   ;Update counter
        SUB     CX,8
        JGE     Mask_Set                ;Modify mask if only one byte
```

```
        NEG     CX
        SHR     BL,CL
        SHL     BL,CL
        XOR     CX,CX                   ;Restore counter
Mask_Set:
        MOV     AL,BL                   ;Fetch mask
        OUT     DX,AL                   ;Set mask register
        MOV     AH,ES:[DI]              ;Latch data
        STOSB                           ;Write new data

        ; Draw pixels from the middle complete bytes

Full:
        MOV     BX,CX                   ;Save count of bits in last byte
        SHR     CX,1                    ;Compute count
        SHR     CX,1
        SHR     CX,1
        JZ      Trail
        MOV     AL,0FFh                 ;Set mask
        OUT     DX,AL
        REP     STOSB

        ; Draw pixels from the trailing partial byte

Trail:
        AND     BL,7
        JZ      End_Scan_Line
        MOV     AL,0FFh                 ;Compute mask
        MOV     CX,BX
        SHR     AL,CL
        NOT     AL
        OUT     DX,AL                   ;Set the mask
        MOV     AL,ES:[DI]              ;Latch data
        STOSB                           ;Set new data

;-----------------------------------------------------------------
; Cleanup and exit
;-----------------------------------------------------------------

End_Scan_Line:
        POP     ES                      ;Restore saved registers
        POP     DS
        POP     SI
        POP     DI
        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
_Scan_Line      ENDP

_TEXT   ENDS
        END
```

# Fill Solid Rectangle

A rectangle is the easiest figure to fill. _Solid_Rect uses the same algorithm described for _Scan_Line, except that the procedure is repeated for a specified number of scan lines with an appropriate page and offset update between successive scan lines.

**Listing 8-5. File: 16COL\RECT.ASM**

```
;************************************************************************
;*                                                                      *
;* File:        RECT.ASM - 4 Bit Planar Solid Rectangle                 *
;* Routine:     _Solid_Rect                                             *
;* Arguments:   X0, Y0, X1, Y1, Color                                   *
;*                                                                      *
;************************************************************************

        INCLUDE VGA.INC

        EXTRN   Select_Color:NEAR
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pitch:WORD
        EXTRN   Graf_Seg:WORD

        PUBLIC  _Solid_Rect

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_X0          EQU     WORD PTR [BP+4] ;Formal parameters
Arg_Y0          EQU     WORD PTR [BP+6]
Arg_X1          EQU     WORD PTR [BP+8]
Arg_Y1          EQU     WORD PTR [BP+10]
Arg_Color       EQU     BYTE PTR [BP+12]

Last_Mask       EQU     BYTE PTR [BP-1] ;Local variables
Start_Page      EQU     BYTE PTR [BP-2]
PageNo          EQU     BYTE PTR [BP-3]

_Solid_Rect     PROC NEAR
        PUSH    BP
        MOV     BP,SP
        SUB     SP,4                    ;Allocate local variables

        PUSH    DI                      ;Preserve registers
        PUSH    SI
        PUSH    DS
        PUSH    ES

;-----------------------------------------------------------------
; Rearange corners so that x0 < x1 and y0 < y1
;-----------------------------------------------------------------

        MOV     AX,Arg_X0               ;Force x0 < x1
        MOV     BX,Arg_X1
        CMP     BX,AX
        JGE     X_In_Order
        MOV     Arg_X0,BX
        MOV     Arg_X1,AX
X_In_Order:
        MOV     AX,Arg_Y0               ;Force y0 < y1
        MOV     BX,Arg_Y1
        CMP     BX,AX
        JGE     Y_In_Order
        MOV     Arg_Y0,BX
        MOV     Arg_Y1,AX
Y_In_Order:
```

```
;----------------------------------------------------------------
; Compute address of first pixel, load set/reset (color) register
;----------------------------------------------------------------

        ; Compute page number and select the page

        MOV     AX,Arg_Y0               ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;       multiply by raster width
        MOV     CX,Arg_X0               ;       add x coordinate/8
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     DI,AX                   ;Save offset within page
        MOV     ES,CS:Graf_Seg          ;Fetch segment
        MOV     AL,DL                   ;Select proper page
        MOV     PageNo,AL               ;Save page number for later
        MOV     Start_Page,AL
        CALL    Select_Page

        ; Load set/reset registers with current color, select bit mask reg

        MOV     AL,Arg_Color
        CALL    Select_Color            ;Also sets DX to gr ctrl data register

        MOV     CX,Arg_X1               ;Compute number of pixels in a line
        SUB     CX,Arg_X0
        INC     CX
        MOV     SI,Arg_Y1               ;Compute number of lines to do
        SUB     SI,Arg_Y0
        INC     SI

;----------------------------------------------------------------
; Draw the rectangle (in three strips = lead, full middle, trail)
;----------------------------------------------------------------

        ; Draw pixels from the leading partial byte

        MOV     AX,Arg_X0               ;Fetch x coordinate
        AND     AX,7                    ;Check for partial byte
        JZ      Full
        XCHG    BH,CL                   ;Preserve counter (CL into BH)
        MOV     BL,0FFh                 ;Compute the mask
        MOV     CL,AL
        SHR     BL,CL
        XCHG    BH,CL                   ;Restore counter
        ADD     CX,AX                   ;Update counter
        SUB     CX,8
        JGE     Mask_Set                ;Modify mask if only one byte
        NEG     CX
        SHR     BL,CL
        SHL     BL,CL
        XOR     CX,CX                   ;Indicate no more bytes
Mask_Set:
        MOV     AL,BL                   ;Fetch mask
        OUT     DX,AL                   ;Set mask register
        PUSH    CX                      ;Preserve counters
        PUSH    DI                      ;Save first byte offset
        MOV     CX,SI                   ;Number of lines to do
        MOV     BX,CS:Video_Pitch
Lead_Loop:
        MOV     AH,ES:[DI]              ;Latch data
        MOV     ES:[DI],AH              ;Write new data
        ADD     DI,BX                   ;Point to next raster
        JC      Lead_Update_Seg
        LOOP    Lead_Loop
        JMP     Lead_Done

Lead_Update_Seg:                        ;Fix page if needed
        XCHG    AL,PageNo               ;Fetch page number (preserve AL)
```

```
        INC     AL                      ;Advance to next page
        CALL    Select_Page
        XCHG    AL,PageNo               ;Save new page number (restore AL)
        LOOP    Lead_Loop

Lead_Done:                              ;Restore counters
        POP     DI
        POP     CX
        INC     DI                      ;Point to first full byte

        ; Draw pixels from the middle complete bytes

Full:
        MOV     Last_Mask,CL            ;Save count of bits in last byte
        AND     Last_Mask,7
        SHR     CX,1                    ;Compute count of full bytes
        SHR     CX,1
        SHR     CX,1
        JCXZ    Trail                   ;Skip if no full bytes
        MOV     BX,CS:Video_Pitch       ;Compute line to line increment
        SUB     BX,CX
        INC     BX
        MOV     AL,Start_Page           ;Restore page number
        MOV     PageNo,AL
        CALL    Select_Page
        MOV     AL,0FFh                 ;Set mask
        OUT     DX,AL
        PUSH    SI
        PUSH    DI

Outer_Loop:
        PUSH    CX                      ;Preserve counter in CX
        REP     STOSB
        DEC     DI                      ;Point to last byte drawn
        ADD     DI,BX                   ;Point to next line
        JC      Full_Update_Seg         ;Check of page crossing
Outer_Update:
        POP     CX                      ;Restore counter (within line)
        DEC     SI                      ;Update counter of lines
        JG      Outer_Loop              ;If not done, go do another line

        POP     DI                      ;Restore pointer
        POP     SI                      ;Restore line counter
        ADD     DI,CX                   ;Point to the trailing byte
        JMP     Trail

Full_Update_Seg:
        XCHG    AL,PageNo               ;Fetch page number (preserve AL)
        INC     AL                      ;Advance to next page
        CALL    Select_Page
        XCHG    AL,PageNo               ;Save new page number (restore AL)
        JMP     Outer_Update

        ; Draw pixels from the trailing partial byte

Trail:
        MOV     CL,Last_Mask            ;Compute number of trailing bits
        OR      CL,CL
        JZ      End_Rect
        MOV     AL,Start_Page           ;Restore page number
        MOV     PageNo,AL
        CALL    Select_Page
        MOV     BX,CS:Video_Pitch       ;Get line to line increment
        MOV     AL,0FFh                 ;Compute mask
        SHR     AL,CL
        NOT     AL
        OUT     DX,AL                   ;Set the mask
        MOV     CX,SI                   ;Counter of bytes to do
Trail_Loop:
        MOV     AL,ES:[DI]              ;Latch data
        MOV     ES:[DI],AL              ;Set new data
```

```
        ADD     DI,BX                    ;Point to next line
        JC      Trail_Update_Seg         ;Update page if needed
        LOOP    Trail_Loop
        JMP     End_Rect

Trail_Update_Seg:
        XCHG    AL,PageNo                ;Fetch page number (preserve AL)
        INC     AL                       ;Advance to next page
        CALL    Select_Page
        XCHG    AL,PageNo                ;Save new page number (restore AL)
        LOOP    Trail_Loop

;-----------------------------------------------------------------
; Clean up and return to caller
;-----------------------------------------------------------------

End_Rect:
        POP     ES                       ;Restore saved registers
        POP     DS
        POP     SI
        POP     DI
        MOV     SP,BP                    ;Restore stack
        POP     BP
        RET
_Solid_Rect     ENDP

_TEXT   ENDS
        END
```

# Clear Screen

A full screen can be filled most efficiently by avoiding all address translations and page boundary detection. _Clear_Screen shows how to efficiently erase the screen. At the start of the procedure, display refresh is disabled to allow data to be moved into display memory at the fastest possible rate. Display refresh normally imposes wait states on the processor when display memory is read or written. Display refresh is re-enabled at the end of the procedure.

**Listing 8-6. File: 16COL\CLEAR.ASM**

```
;************************************************************************
;*                                                                      *
;* File:        CLEAR.ASM - 4 Bit Planar Pixel Clear Screen             *
;* Routine:     _Clear_Screen                                           *
;* Arguments:   Color                                                   *
;*                                                                      *
;************************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pages:WORD
        EXTRN   Select_Color:NEAR

        PUBLIC  _Clear_Screen

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_Color       EQU     BYTE PTR [BP+4]

_Clear_Screen   PROC NEAR
        PUSH    BP                       ;Standard high-level entry
        MOV     BP,SP
```

```
        PUSH    ES                      ;Preserve registers
        PUSH    DI

        ; Enable maximum access to display memory (disable video refresh)

        MOV     DX,SEQUENCER_PORT       ;Fetch address of sequencer
        MOV     AL,1                    ;Index of clock select register
        OUT     DX,AL                   ;Select register
        INC     DX
        IN      AL,DX                   ;Read current value (to be restored)
        MOV     AH,AL                   ;Save current value
        OR      AL,20h                  ;Set disable video bit
        OUT     DX,AL                   ;Disable video refresh
        MOV     AL,1
        PUSH    AX                      ;Save old value for later

        ; Clear display memory

        MOV     AL,Arg_Color            ;Color to fill with
        CALL    Select_Color
        MOV     AL,0FFh                 ;Enable 8 bits for write
        OUT     DX,AL
        MOV     ES,CS:Graf_Seg          ;Select first segment
        XOR     AX,AX                   ;Initialize page counter

Cls_Page_Loop:
        CALL    Select_Page             ;Select next page
        XOR     DI,DI                   ;Set offset
        MOV     CX,8000h                ;Number of words to clear
        REP     STOSW                   ;Clear the next segment
        INC     AX                      ;Update page counter
        CMP     AX,CS:Video_Pages       ;All pages cleared?
        JL      Cls_Page_Loop           ;If not go clear next one

        ; Restore video refresh

        MOV     DX,SEQUENCER_PORT       ;Fetch address of sequencer
        POP     AX                      ;Fetch previous value
        OUT     DX,AX                   ;Restore

        POP     DI                      ;Restore registers
        POP     ES
        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
_Clear_Screen   ENDP

_TEXT   ENDS
        END
```

## Copy Block

_BitBlt shows how to perform simple block copying where both the source and destination are in display memory. The dual page capability of some VGA boards could be used to improve performance. The module BITBLT.ASM is divided into two sections according to direction of traversal, which is determined by overlaps between the source and destination rectangle. Except for movement between scan lines, the sections are identical. Each scan line is transferred using the following steps:

Convert starting x,y addresses to Page:Offset
Call procedure Compute_Phase to get phase and masks
Loop over scan lines, and call procedure Copy_Raster to do following:

Get enough bits from source (from one or two bytes) for one destination byte
Rotate bits into place (using phase computed earlier)
Move first byte into intermediate buffer (first partial byte for destination)
Loop constructing intermediate bytes as follows:
    Fetch source
    Rotate into place
    Save left-over bits (rotated out) for next loop
    Combine with left-over from previous loop
    Move to next byte in intermediate buffer
Setup mask for first partial byte of destination
Latch destination byte, then write first byte from intermediate buffer
Setup mask to write all eight bits
Copy rest of the bytes from intermediate buffer to destination
Setup mask for last partial byte
Latch destination byte, then write last byte from intermediate buffer

Each raster is first aligned and copied into intermediate buffer, and then the intermediate buffer is copied into destination (properly masking off the first and last partial bytes).

A significant performance improvement can be obtained for cases where the source is byte-aligned with the destination (phase = 0), or when the VGA board is capable of two separate pages. In such cases latched write mode 1 can be used with the MOVSB instruction to transfer data for each scan line.

**Listing 8-7. File: 16COL\BITBLT.ASM**

```
;************************************************************************
;*                                                                      *
;* File:        BITBLT.ASM - 4 Bit Planar Bit Block Transfer            *
;* Routine:     _BitBlt                                                 *
;* Arguments:   Source X, Source Y, Destination X, Destination Y,       *
;*              Width, Height, Logical function for function register  *
;*                                                                      *
;************************************************************************

        INCLUDE VGA.INC

        EXTRN   Select_Page:NEAR
        EXTRN   Graf_Seg:WORD
        EXTRN   Video_Pitch:WORD
        EXTRN   Ras_Buffer:BYTE

        PUBLIC  _BitBlt

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_Src_X       EQU     WORD PTR [BP+4] ;Formal parameters
Arg_Src_Y       EQU     WORD PTR [BP+6]
Arg_Dst_X       EQU     WORD PTR [BP+8]
Arg_Dst_Y       EQU     WORD PTR [BP+10]
Arg_DX          EQU     WORD PTR [BP+12]
Arg_DY          EQU     WORD PTR [BP+14]
Arg_Fn          EQU     BYTE PTR [BP+16]
```

```
Src_Pitch       EQU     WORD PTR [BP-02];Local variables
Dst_Pitch       EQU     WORD PTR [BP-04]
First_Mask      EQU     BYTE PTR [BP-05]
Last_Mask       EQU     BYTE PTR [BP-06]
Phase           EQU     BYTE PTR [BP-07]
First_Fetch     EQU     BYTE PTR [BP-08]
Full_Count      EQU     BYTE PTR [BP-09]
Plane           EQU     BYTE PTR [BP-10]
Byte_Count      EQU     WORD PTR [BP-12]
SrcPage         EQU     BYTE PTR [BP-13]
DstPage         EQU     BYTE PTR [BP-14]
FirstSrc        EQU     BYTE PTR [BP-15]
FirstDst        EQU     BYTE PTR [BP-16]

_BitBlt PROC    NEAR
        PUSH    BP
        MOV     BP,SP
        SUB     SP,16                   ;Allocate space for local variables

        PUSH    DS                      ;Preserve segment registers
        PUSH    ES
        PUSH    DI
        PUSH    SI

        ; Set source and destination pitch

        MOV     CX,CS:Video_Pitch       ;Fetch screen width
        MOV     Src_Pitch,CX            ;Set src and dst pitch
        MOV     Dst_Pitch,CX

        CALL    Reset_Graphics_Controller ;Put gr ctrl in known state

        ; Set logical function

        MOV     DX,GRAPHICS_CTRL_PORT   ;Select data rotate and Arg_Fn register
        MOV     AL,Fn_SEL_REG
        MOV     AH,Arg_Fn               ;Force function into 0-3 and rotate
        AND     AH,3                    ;into bits 3,4
        SHL     AH,1
        SHL     AH,1
        SHL     AH,1
        OUT     DX,AX                   ;Set

        ;-----------------------------------------------------------------
        ; Determine direction of traversal
        ; (Note that a check for x reversal is not needed since an
        ; intermediate buffer is being used for transfer)
        ;-----------------------------------------------------------------

        MOV     AX,Arg_Dst_Y                    ;Check dsty,srcy
        CMP     AX,Arg_Src_Y
        JL      BB_XPYP
        JMP     BB_XPYN

        ;-----------------------------------------------------------------
        ; Traverse x+ y+
        ;-----------------------------------------------------------------

BB_XPYP:

        ; Compute src and dst address

        MOV     AX,Arg_Src_Y            ;Convert x,y to Page:Offset
        MUL     CS:Video_Pitch
        MOV     CX,Arg_Src_X
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     DS,CS:Graf_Seg          ;Put address in DS:SI
```

```
        MOV     SI,AX
        MOV     SrcPage,DL              ;Save page number
        MOV     FirstSrc,DL
        MOV     AX,Arg_Dst_Y            ;Convert x,y to Page:Offset
        MUL     CS:Video_Pitch
        MOV     CX,Arg_Dst_X
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     ES,CS:Graf_Seg          ;Put address in ES:DI
        MOV     DI,AX
        MOV     DstPage,DL              ;Save page number
        MOV     FirstDst,DL

        ; Compute phase and masks

        CALL    Compute_Phase           ;Get alignment info, masks,
        MOV     First_Mask,AL           ;Save masks
        MOV     Last_Mask,AH
        MOV     Phase,CL                ;Save phase
        MOV     First_Fetch,BH          ;Save number of bytes for first byte
        MOV     Full_Count,CH           ;Save number of full bytes

        ; Loop over planes to be copied

        MOV     Plane,3                 ;Number of planes to do
XPYP_Plane_Loop:
        MOV     DL,Plane                ;Fetch next plane to do
        CALL    Select_Plane            ;Select plane for read and write
        MOV     AL,FirstSrc             ;Reset page numbers
        MOV     SrcPage,AL
        MOV     AL,FirstDst
        MOV     DstPage,AL

        PUSH    SI                      ;Preserve starting addresses
        PUSH    DI
        MOV     BX,Arg_DY               ;Number of rasters to transfer

XPYP_Raster_Loop:
        PUSH    BX                      ;Preserve raster counter
        CALL    Copy_Raster             ;Copy the block
        POP     BX                      ;Pointers to current raster
        DEC     BX                      ;Update number of rasters to do
        JLE     XPYP_RasterDone
        ADD     SI,Src_Pitch            ;Update pointer to point to next raster
        JNC     XPYP_NUSPage            ;Update page on carry
        INC     SrcPage                 ;Update page number
XPYP_NUSPage:
        ADD     DI,Dst_Pitch
        JNC     XPYP_NUDPage            ;Update page on carry
        INC     DstPage                 ;Update destination page number
XPYP_NUDPage:
        JMP     XPYP_Raster_Loop        ;And repeat if not all done
XPYP_RasterDone:

        POP     DI                      ;Restore starting addresses
        POP     SI

        DEC     Plane                   ;Update number of planes to do
        JGE     XPYP_Plane_Loop         ;And do next plane if any left to do
        JMP     End_BitBlt

        ;-------------------------------------------------------------
        ; Traverse x+ y-
        ;-------------------------------------------------------------

BB_XPYN:

        ; Compute src and dst addresses
```

```
        MOV     AX,Arg_Src_Y            ;Convert x,y to Page:Offset
        ADD     AX,Arg_DY
        DEC     AX
        MOV     CX,Arg_Src_X
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        MUL     CS:Video_Pitch
        ADD     AX,CX
        ADC     DX,0
        MOV     DS,CS:Graf_Seg          ;Put address in DS:SI
        MOV     SI,AX
        MOV     SrcPage,DL
        MOV     FirstSrc,DL

        MOV     AX,Arg_Dst_Y            ;Convert x,y to Page:Offset
        ADD     AX,Arg_DY
        DEC     AX
        MOV     CX,Arg_Dst_X
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        MUL     CS:Video_Pitch
        ADD     AX,CX
        ADC     DX,0
        MOV     ES,CS:Graf_Seg          ;Put address in ES:DI
        MOV     DI,AX
        MOV     DstPage,DL
        MOV     FirstDst,DL

        ; Compute phase and masks

        CALL    Compute_Phase           ;Get alignment info, masks,
        MOV     First_Mask,AL           ;Save masks
        MOV     Last_Mask,AH
        MOV     Phase,CL                ;Save phase
        MOV     First_Fetch,BH          ;Save number of bytes for first byte
        MOV     Full_Count,CH           ;Save number of full bytes

        ; Loop over planes to be copied

        MOV     Plane,3                 ;Number of planes to do
XPYN_Plane_Loop:
        MOV     DL,Plane                ;Fetch next plane to do
        CALL    Select_Plane            ;Select plane for read and write
        MOV     AL,FirstSrc             ;Reset page numbers
        MOV     SrcPage,AL
        MOV     AL,FirstDst
        MOV     DstPage,AL

        PUSH    SI                      ;Preserve starting addresses
        PUSH    DI
        MOV     BX,Arg_DY               ;Number of rasters to transfer

XPYN_Raster_Loop:
        PUSH    BX                      ;Preserve raster counter
        CALL    Copy_Raster             ;Copy the block
        POP     BX                      ;Pointers to current raster
        DEC     BX                      ;Update number of rasters to do
        JLE     XPYN_RasterDone
        SUB     SI,Src_Pitch            ;Update pointer to point to next raster
        JNC     XPYN_NUSPage            ;Update page on carry
        DEC     SrcPage                 ;Update page number
XPYN_NUSPage:
        SUB     DI,Dst_Pitch
        JNC     XPYN_NUDPage            ;Update page on carry
        DEC     DstPage                 ;Update destination page number
XPYN_NUDPage:
        JMP     XPYN_Raster_Loop        ;And repeat if not all done
XPYN_RasterDone:
```

```
        POP     DI                      ;Restore starting addresses
        POP     SI
        DEC     Plane                   ;Update number of planes to do
        JGE     XPYN_Plane_Loop         ;And do next plane if any left to do
        JMP     End_BitBlt

        ;-----------------------------------------------------------
        ; Cleanup and return
        ;-----------------------------------------------------------

End_BitBlt:
        CALL    Reset_Graphics_Controller

        POP     SI                      ;Restore segment registers
        POP     DI
        POP     ES
        POP     DS
        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
_BitBlt ENDP

;*********************************************************************
;                                                                    *
; Compute_Phase                                                      *
;       Compute alignment and masks for blit when                    *
;                                                                    *
; Entry: Src_X, Src_Y                                                *
;        Dst_X, Dst_Y                                                *
;        DX   , DY            Blit description passed via [BP+4]...  *
;                                                                    *
; Exit: AL - Mask for first byte                                     *
;       AH - Mask for last byte (or 0 if not needed)                 *
;       CL - Phase alignment                                         *
;       CH - Number of "full" bytes                                  *
;       BH - Number of fetches needed for first byte                 *
;                                                                    *
;*********************************************************************

Compute_Phase   PROC NEAR

        ; Compute masks for first and last byte

        MOV     CX,Arg_Dst_X            ;Number of clear bits in first byte
        AND     CL,7
        MOV     AL,0FFh                 ;Compute mask to clear leading bits
        SHR     AL,CL
        MOV     BL,8                    ;Save number of bits in first byte
        SUB     BL,CL
        XOR     BH,BH

        MOV     CX,Arg_Dst_X            ;Get address of last bit
        ADD     CX,Arg_DX
        AND     CL,7                    ;Position just after last bit
        MOV     AH,0FFh                 ;Compute mask to keep bits after
        SHR     AH,CL
        NOT     AH                      ;Complement to keep leading bits

        ; Combine masks if all bits in the same byte

        CMP     BX,Arg_DX               ;Check if first byte has all the bits
        JLE     Masks_Done              ;Jump if not
        AND     AL,AH                   ;Combine masks,
        XOR     AH,AH                   ;indicate no last byte,
        MOV     BX,Arg_DX               ;and adjust BX to have the actual number
Masks_Done:

        ;Compute phase alignment

        MOV     CX,Arg_Dst_X            ;Compute bit distance between
        SUB     CX,Arg_Src_X            ;bit positions in first bytes
```

```
        AND     CL,7                    ;of source and destination bytes
                                        ;as (DST-SRC)&7

        ; Compute number of "full" bytes

        NEG     BX                      ;Negate number of bits in first byte
        ADD     BX,Arg_DX               ;Add total number of bits to trasfer
        SHR     BX,1                    ;Convert bit count to byte count
        SHR     BX,1
        SHR     BX,1
        MOV     CH,BL                   ;Copy count into CH

        ;Compute number of fetches needed for first byte

        MOV     BX,Arg_Src_X            ;Check if need one or two bytes
        AND     BL,7                    ;for first byte of destination
        MOV     DX,Arg_Dst_X            ;If (src mod 7) > (dst mod 7) then 2
        AND     DL,7
        CMP     BL,DL
        MOV     BH,1                    ;Assume one fetch
        JLE     Fetches                 ;Jump if only one fetch needed
        INC     BH                      ;Must use two fetches
Fetches:
        RET
Compute_Phase   ENDP

;************************************************************************
;                                                                       *
; Copy_Raster                                                           *
;       Transfer pixels from one raster. The transfer is done in the    *
;       following steps:                                                *
;                                                                       *
;         (1) LEADING PARTIAL BYTE FOR DESTINATION IS PLACED IN TMP BUF*
;         (2) FULL BYTES OF DESTINATION ARE MOVED INTO TEMP BUFFER     *
;         (3) TRAILING PARTIAL BYTE FOR DESTINATION IS PLACED IN TMP   *
;         (4) TEMPORARY BUFFER IS COPIED INTO DESTINATION              *
; Entry:                                                                *
;       First_Mask      EQU     BYTE PTR [BP-05]                        *
;       Last_Mask       EQU     BYTE PTR [BP-06]                        *
;       First_Fetch     EQU     BYTE PTR [BP-08]                        *
;       Full_Count      EQU     BYTE PTR [BP-09]                        *
;       DS:SI           Source                                          *
;       ES:DI           Destination                                     *
;       BX              Number of lines to transfer                     *
;       CL              Phase                                           *
;                                                                       *
;************************************************************************

Copy_Raster     PROC NEAR

        PUSH    SI                      ;Preserve pointers
        PUSH    DI

        ;-------------------------------------------------------------
        ; Construct data for destination in a temporary buffer
        ;-------------------------------------------------------------

        PUSH    ES                      ;Setup ES:DI to point to temp buff
        PUSH    DI
        LEA     DI,CS:Ras_Buffer
        MOV     AX,CS
        MOV     ES,AX
        MOV     AL,SrcPage              ;Select source page
        CALL    Select_Page

        ; Get enough bits from source to construct first partial byte of dst

        MOV     CL,Phase                ;Fetch phase alignment
        CMP     First_Fetch,1           ;Check if need one or two bytes of src
        JLE     Get_1                   ;for first byte of destination
Get_2:                                  ;Get next byte of source
```

```
        LODSB                           ;Fetch first source byte
        MOV     AH,AL                   ;Place into 'left over' byte

Get_1:
        LODSB                           ;Get next byte of source
        MOV     BH,AL                   ;Save for later
        ROR     AX,CL                   ;Rotate into place
        MOV     AH,BH                   ;Put unused bits into AH
        STOSB                           ;Set destination

        ; Loop over complete bytes within a single source raster

        MOV     BL,Full_Count           ;Number of bytes to copy
        XOR     BH,BH
        OR      BX,BX
        JZ      Last                    ;Skip if none
Full_Loop:
        LODSB                           ;Fetch next source byte
        MOV     DH,AL                   ;Save unused bits for later
        SHR     AX,CL                   ;Align
        MOV     AH,DH                   ;Keep unused bits
        STOSB                           ;Save the byte in destination
        DEC     BX
        JG      Full_Loop

        ; Construct the final partial byte

Last:
        LODSB                           ;Fetch next byte of source
        SHR     AX,CL                   ;Align with destination
        STOSB                           ;Set destination

        ;-----------------------------------------------------------
        ; Copy data from temporary buffer to destination
        ;-----------------------------------------------------------

        POP     DI                      ;Restore pointers
        POP     ES
        LEA     SI,CS:Ras_Buffer        ;Setup DS:SI to temp buffer
        MOV     AX,CS
        MOV     DS,AX
        MOV     AL,DstPage              ;Select destination page
        CALL    Select_Page

        ; Move first leading partial byte from temp buffer to destination

        MOV     DX,GRAPHICS_CTRL_PORT   ;Set write mask to First_Mask
        MOV     AL,BIT_MASK_REG
        MOV     AH,First_Mask
        OUT     DX,AX

        MOV     AH,ES:[DI]              ;Latch destination byte
        MOVSB                           ;Set first partial byte

        ; Move middle complete bytes from temp buffer to destination

        MOV     AH,0FFh                 ;Reset write mask to all bits
        OUT     DX,AX

        MOV     CL,Full_Count           ;Number of bytes to move
        XOR     CH,CH
        JCXZ    FullDone
        CMP     Arg_Fn,0                ;Use MOVS if function is 0 (SRC)
        JZ      FnSrc
FullLoop:
        MOV     AH,ES:[DI]              ;Latch destination
        MOVSB                           ;Combine source and latch into dst
        LOOP    FullLoop
        JMP     FullDone
FnSrc:
        REP     MOVSB                   ;Move data from temporary buf to dst
```

```
FullDone:

        ; Move the last partial byte from temporary buffer to destination

        MOV     AH,Last_Mask            ;Set write mask to Last_Mask
        OUT     DX,AX

        MOV     AL,ES:[DI]              ;Latch data
        MOVSB                           ;Write new data

        MOV     DS,CS:Graf_Seg          ;Restore pointers
        POP     DI
        POP     SI

        RET
Copy_Raster     ENDP

;************************************************************************
;                                                                       *
; Reset_Graphics_Controller                                             *
;       Restore registers modified by bitblt routines                   *
;                                                                       *
;************************************************************************

Reset_Graphics_Controller PROC NEAR
        MOV     DX,GRAPHICS_CTRL_PORT   ;Set write mask to all bits
        MOV     AL,BIT_MASK_REG
        MOV     AH,0FFh
        OUT     DX,AX

        MOV     AL,Fn_SEL_REG           ;Set function to direct write
        MOV     AH,0
        OUT     DX,AX

        MOV     AL,READ_PLANE_REG       ;Select read plane to 0
        OUT     DX,AX

        MOV     AL,SR_ENABLE_REG        ;Disable set/rest function
        OUT     DX,AX

        MOV     DX,SEQUENCER_PORT       ;Fetch register port
        MOV     AL,PLANE_ENABLE_REG     ;Fetch register index
        MOV     AH,0Fh                  ;Enable all planes for write
        OUT     DX,AX                   ;Select plane for write
        RET
Reset_Graphics_Controller ENDP

;************************************************************************
;                                                                       *
; Select_Plane                                                          *
;       Select plane passed in DL for read and write                    *
;                                                                       *
; Entry: DL - Page number                                               *
;                                                                       *
;************************************************************************

Select_Plane    PROC NEAR
        PUSH    DX
        MOV     AH,DL
        ;Enable read
        AND     AH,3                    ;Force into range
        MOV     DX,GRAPHICS_CTRL_PORT   ;Fetch register port
        MOV     AL,READ_PLANE_REG       ;Fetch register index
        OUT     DX,AX                   ;Select plane for read
        ;Enable write
        XCHG    AL,CL                   ;Preserve CL
        MOV     CL,AH                   ;Convert plane number
        MOV     AH,1                    ;  to bit position
        SHL     AH,CL
        XCHG    AL,CL                   ;Restore CL
        MOV     DX,SEQUENCER_PORT       ;Fetch register port
```

```
        MOV     AL,PLANE_ENABLE_REG      ;Fetch register index
        OUT     DX,AX                    ;Select plane for write
        POP     DX
        RET
Select_Plane    ENDP

_TEXT   ENDS
        END
```

## Set Cursor, Move Cursor, Remove Cursor

This module contains three procedures to define, move, and remove a cursor in the display memory.

In the procedure _Set_Cursor, monochrome XOR and AND masks are expanded according to the parameters FG_Color (foreground color) and BG_Color (background color). In this implementation these masks are stored on screen in an area immediately below the first scan line in order to clearly see how the cursor is constructed. By changing one line of marked code, the cursor mask storage area can be moved offscreen. The entire cursor mask storage area must reside within one page of display memory.

At the end of the _set_Cursor procedure, the variables Last_Cursor_X and Last_Cursor_Y are initialized to ensure proper operation during the first call to _Move_Cursor.

In the procedure _Move_Cursor, the cursor masks are logically combined with the background data from the new cursor position specified. A block twice the size of the cursor is used to minimize flicker for small changes in cursor position. Background data for a block around the cursor position is kept immediately next to the cursor masks. A check is made to see if the cursor moved outside of the current block, and if so, the cursor is removed from the screen (by calling _Remove_Cursor) and a new block is copied to the save area. Next, the background save area is copied into the build area (next to the save area), where the cursor masks are combined with the background data. The data in the build area is then copied to the display.

For a small motion of the cursor (within the same block), the cursor in the display area is removed and placed in its new position in a single transfer; the cursor never disappears from the screen and flicker is eliminated (until an edge of the block is reached).

_Remove_Cursor restores the area under the cursor by transferring data from the save area to the display.

**Listing 8-8. File: 16COL\CURSOR.ASM**

```
;*************************************************************************
;*                                                                       *
;* File:  CURSOR.ASM - 4 Bit Planar Cursor Routines                      *
;* Routines:   _Set_Cursor, _Move_Cursor, _Remove_Cursor                 *
;*                                                                       *
;*************************************************************************

    INCLUDE VGA.INC
```

```
     EXTRN      _BitBlt:NEAR
     EXTRN      Select_Color:NEAR

     EXTRN      Graf_Seg:WORD
     EXTRN      Video_Pitch:WORD
     EXTRN      Video_Height:WORD
     EXTRN      Select_Page:NEAR
     EXTRN      Select_Write_Page:NEAR
     EXTRN      Select_Read_Page:NEAR
     EXTRN      Two_Pages:BYTE

     PUBLIC     _Set_Cursor
     PUBLIC     _Move_Cursor
     PUBLIC     _Remove_Cursor

_TEXT     SEGMENT BYTE PUBLIC 'CODE'

;-----------------------------------------------------------------------
; Common cursor definitions
;-----------------------------------------------------------------------

CUR_WIDTH         EQU     32
CUR_HEIGHT        EQU     32

Save_Area_Off     DW      0
Save_Area_Page    DB      0
Save_Area_x       DW      2*CUR_WIDTH
AND_Area_y        LABEL   WORD
XOR_Area_y        LABEL   WORD
Build_Area_y      LABEL   WORD
Save_Area_y       DW      0
Build_Area_x      DW      4*CUR_WIDTH
AND_Area_x        DW      0
XOR_Area_x        DW      CUR_WIDTH
Last_Cursor_x     DW      0
Last_Cursor_y     DW      0

;***********************************************************************
;*                                                                     *
;* _Set_Cursor                                                         *
;*   This procedure will expand the two cursor masks into              *
;*   four planes.  Normally the masks should be stored after the       *
;*   last visible scan line (global parameter 'Video_Height',          *
;*   however in this demo, the cursor masks and the 'save buffer'      *
;*   will be stored immediately above the last line.  This is done     *
;*   so that the reader can clearly see the AND mask, the XOR mask,    *
;*   and the area under the cursor in 'save buffer'.                   *
;*                                                                     *
;* Entry:                                                              *
;*   AND_Mask - 4x32 bytes with AND mask                               *
;*   XOR_Mask - 4x32 bytes with XOR mask                               *
;*   BG_Color - Foreground color                                       *
;*   FG_Color - Background color                                       *
;*                                                                     *
;***********************************************************************

Arg_AND_Mask    EQU  WORD PTR [BP+4] ;Formal parameters
Arg_XOR_Mask    EQU  WORD PTR [BP+6]
Arg_BG_Color    EQU  BYTE PTR [BP+8]
Arg_FG_Color    EQU  BYTE PTR [BP+10]

_Set_Cursor     PROC NEAR
     PUSH       BP                    ;Standard high-level entry
     MOV        BP,SP

     PUSH       SI                    ;Save registers
     PUSH       DI
     PUSH       ES
     PUSH       DS

     ; Setup Graphics Controller to use 'Set/Reset' mode with
```

```
     ; Set/Reset value set to background color

     MOV  AL,Arg_BG_Color      ;Select background color
     CALL Select_Color
     MOV  AL,0FFh              ;Enable 8 bits for write
     OUT  DX,AL                ;(Select_Color selected register)

     ; Fill with background, the area where both masks will be stored
     ; (Note that this will work only if save area is contained in one page)

     MOV  CX,0                 ;Set x to start of save area
     MOV  AX,CS:Video_Height   ;Set y to below last line on the screen
     ;!!!!!!!!! The next line should be removed    !!!!!!!!!!!!!!!!
     ;!!!!!!!!! if you do not want to see the save !!!!!!!!!!!!!!!!
     ;!!!!!!!!! regions on the screen              !!!!!!!!!!!!!!!!
     MOV  AX,0                 ;Make visible for demo !!!!!!!!!!!!!!
     MOV  CS:Save_Area_y,AX    ;Save y for other cursor procs
     MUL  CS:Video_Pitch       ;Convert x,y to Seg:Off
     MOV  CS:Save_Area_Off,AX  ;Setup Seg:Off for other routines
     MOV  DI,AX                ;Set DI to save area offset
     MOV  ES,CS:Graf_Seg       ;Point ES to save area segment
     MOV  AL,DL                ;Select page
     CALL Select_Page
     MOV  CX,CUR_HEIGHT        ;Number of scanlines to do
     MOV  BX,CS:Video_Pitch    ;Calculate scan-to-scan increment
     SUB  BX,2*CUR_WIDTH/8
Fill_Background:
     STOSW                     ;Bits for AND mask
     STOSW
     STOSW                     ;Bits for XOR mask
     STOSW
     ADD  DI,BX                ;Point to next scanline
     LOOP Fill_Background

     ; Set foreground bits for the AND mask in the save area

     MOV  AL,Arg_FG_Color      ;Load Set/Reset with foreground color
     CALL Select_Color         ;Bit mask register now selected

     MOV  CX,CUR_HEIGHT        ;Initialize counter
     MOV  DI,CS:Save_Area_Off  ;Get pointer to save area
     MOV  SI,Arg_AND_Mask      ;Advance pointer to AND-mask section
     ADD  BX,CUR_WIDTH/8       ;Set scan-to-scan increment
Set_AND_FG:
     LODSB                     ;Fetch next byte from the mask
     OUT  DX,AL                ;Set bit mask register using cursor mask
     MOV  AH,ES:[DI]           ;Latch data
     STOSB                     ;Write next 8 bits
     LODSB                     ;Fetch next byte from the mask
     OUT  DX,AL                ;Set bit mask register using cursor mask
     MOV  AH,ES:[DI]           ;Latch data
     STOSB                     ;Write next 8 bits
     LODSB                     ;Fetch next byte from the mask
     OUT  DX,AL                ;Set bit mask register using cursor mask
     MOV  AH,ES:[DI]           ;Latch data
     STOSB                     ;Write next 8 bits
     LODSB                     ;Fetch next byte from the mask
     OUT  DX,AL                ;Set bit mask register using cursor mask
     MOV  AH,ES:[DI]           ;Latch data
     STOSB                     ;Write next 8 bits
     ADD  DI,BX
     LOOP Set_AND_FG

     ; Change foreground bits for the XOR mask in the save area

     MOV  CX,CUR_HEIGHT        ;Initialize counter
     MOV  DI,CS:Save_Area_Off  ;Fetch pointer to save area
     ADD  DI,CUR_WIDTH/8       ;Advance pointer to XOR-mask section
     MOV  SI,Arg_XOR_Mask      ;Get pointer to XOR mask
Set_XOR_FG:
     LODSB                     ;Fetch next byte from the mask
```

```
     OUT  DX,AL               ;Set bit mask register using cursor mask
     MOV  AH,ES:[DI]          ;Latch data
     STOSB                    ;Write next 8 bits
     LODSB                    ;Fetch next byte from the mask
     OUT  DX,AL               ;Set bit mask register using cursor mask
     MOV  AH,ES:[DI]          ;Latch data
     STOSB                    ;Write next 8 bits
     LODSB                    ;Fetch next byte from the mask
     OUT  DX,AL               ;Set bit mask register using cursor mask
     MOV  AH,ES:[DI]          ;Latch data
     STOSB                    ;Write next 8 bits
     LODSB                    ;Fetch next byte from the mask
     OUT  DX,AL               ;Set bit mask register using cursor mask
     MOV  AH,ES:[DI]          ;Latch data
     STOSB                    ;Write next 8 bits
     ADD  DI,BX
     LOOP Set_XOR_FG

     ; Save 'previous cursor' to save area (this is needed for first
     ; call to Move_Cursor procedure, because it always restores and
     ; we need meaningful data for the first restore)

     MOV  AX,CS:Save_Area_x   ;Use save area as last cursor pos
     MOV  CS:Last_Cursor_x,AX
     MOV  AX,CS:Save_Area_y
     MOV  CS:Last_Cursor_y,AX ;Save 'where it came from'

     ; Clean up and return

     POP  DS                  ;Restore segment registers
     POP  ES
     POP  DI
     POP  SI

     MOV  SP,BP               ;Restore stack
     POP  BP
     RET
_Set_Cursor    ENDP

;***********************************************************************
;*                                                                     *
;* _Move_Cursor                                                        *
;*   This procedure is used to move the cursor from one                *
;*   location to another. The cursor move is performed using the       *
;*   following steps:                                                  *
;*        1 - Check if new cursor is outside 'cursor block'            *
;*        2 - If outside 'cursor block' restore area under             *
;*            previous block.                                          *
;*            Save area under new block.                               *
;*        3 - Copy saved are into cursor build area (both save and     *
;*            build areas are normally off-screen).                    *
;*        4 - Combine AND and XOR masks with build area.               *
;*        5 - Copy build area to where new cursor should be (this      *
;*            in most cases overwrites the old cursor).                *
;*   The 'build area' is a rectangle twice the size of the cursor.     *
;*   It is used to eliminate flicker for small movement of the         *
;*   cursor, since cursor may not need to be erased if it moves        *
;*   only by a few pixels.                                             *
;*                                                                     *
;* Entry:                                                              *
;*   Curs_X - Position of the new cursor                               *
;*   Curs_Y                                                            *
;*                                                                     *
;***********************************************************************

Arg_Curs_x     EQU  WORD PTR [BP+4] ;Formal parameters
Arg_Curs_y     EQU  WORD PTR [BP+6]

_Move_Cursor   PROC NEAR
     PUSH BP                        ;Standard high-level entry
     MOV  BP,SP
```

```
        PUSH SI                         ;Save registers
        PUSH DI
        PUSH ES
        PUSH DS

        ; Check if new area needs to be saved

        MOV  AX,Arg_Curs_x        ;Fetch new x
        AND  AX,NOT(CUR_WIDTH-1)  ;Round to nearest buffer block
        MOV  BX,Arg_Curs_y        ;Fetch new y
        AND  BX,NOT(CUR_HEIGHT-1) ;Round to nearest buffer block

        CMP  AX,CS:Last_Cursor_x  ;Check if x moved into next block
        JNE  Cursor_New_Block
        CMP  BX,CS:Last_Cursor_y  ;Check if y moved into next block
        JNE  Cursor_New_Block
        JMP  Build_Cursor

        ; For new block, call to remove old cursor, then use _BitBlt
        ; to save block under next cursor location into the save area

Cursor_New_Block:
        CALL _Remove_Cursor       ;Restore last location
        MOV  AX,Arg_Curs_x        ;Fetch new x
        AND  AX,NOT(CUR_WIDTH-1)  ;Round to nearest buffer block
        MOV  CS:Last_Cursor_x,AX  ;Save as 'last x'
        MOV  AX,Arg_Curs_y        ;Fetch new y
        AND  AX,NOT(CUR_HEIGHT-1) ;Round to nearest buffer block
        MOV  CS:Last_Cursor_y,AX  ;Save as 'last y'

        MOV  AX,FUNC_COPY         ;Push function on the stack
        PUSH AX
        MOV  AX,2*CUR_HEIGHT      ;Push width and height
        PUSH AX
        MOV  AX,2*CUR_WIDTH
        PUSH AX
        PUSH CS:Save_Area_y       ;Push x and y of destination
        PUSH CS:Save_Area_x
        PUSH CS:Last_Cursor_y     ;Push x and y of source
        PUSH CS:Last_Cursor_x
        CALL _BitBlt
        ADD  SP,14

        ; Use _BitBlt to copy save area into build area

Build_Cursor:
        MOV  AX,FUNC_COPY         ;Push function on the stack
        PUSH AX
        MOV  AX,2*CUR_HEIGHT      ;Push width and height
        PUSH AX
        MOV  AX,2*CUR_WIDTH
        PUSH AX
        PUSH CS:Build_Area_y      ;Push x and y of destination
        PUSH CS:Build_Area_x
        PUSH CS:Save_Area_y       ;Push x and y of source
        PUSH CS:Save_Area_x
        CALL _BitBlt
        ADD  SP,14

        ; Use _BitBlt procedure to mix AND and XOR masks of the cursor
        ; with build area

        MOV  AX,FUNC_AND          ;Push function on the stack
        PUSH AX
        MOV  AX,CUR_HEIGHT        ;Push width and height
        PUSH AX
        MOV  AX,CUR_WIDTH
        PUSH AX
        MOV  AX,Arg_Curs_y        ;Compute offset of new cursor in block
        AND  AX,CUR_HEIGHT-1
        ADD  AX,CS:Build_Area_y   ;Add location of build block
```

```
     PUSH AX
     MOV  AX,Arg_Curs_x        ;Compute offset of new cursor in block
     AND  AX,CUR_WIDTH-1
     ADD  AX,CS:Build_Area_x   ;Add location of build block
     PUSH AX
     PUSH CS:AND_Area_y        ;Push x and y of source
     PUSH CS:AND_Area_x
     CALL _BitBlt
     ADD  SP,14

     MOV  AX,FUNC_XOR          ;Push function on the stack
     PUSH AX
     MOV  AX,CUR_HEIGHT        ;Push width and height
     PUSH AX
     MOV  AX,CUR_WIDTH
     PUSH AX
     MOV  AX,Arg_Curs_y        ;Compute offset of new cursor in block
     AND  AX,CUR_HEIGHT-1
     ADD  AX,CS:Build_Area_y   ;Add location of build block
     PUSH AX
     MOV  AX,Arg_Curs_x        ;Compute offset of new cursor in block
     AND  AX,CUR_WIDTH-1
     ADD  AX,CS:Build_Area_x   ;Add location of build block
     PUSH AX
     PUSH CS:XOR_Area_y        ;Push x and y of source
     PUSH CS:XOR_Area_x
     CALL _BitBlt
     ADD  SP,14

     ; Use _BitBlt procedure to copy build area to screen (and erase old
     ; cursor with the new cursor block).

     MOV  AX,FUNC_COPY         ;Push function to use
     PUSH AX
     MOV  AX,2*CUR_HEIGHT      ;Push width and height
     PUSH AX
     MOV  AX,2*CUR_WIDTH
     PUSH AX
     MOV  AX,Arg_Curs_y        ;Push x and y of destination
     AND  AX,NOT(CUR_HEIGHT-1)
     PUSH AX
     MOV  AX,Arg_Curs_x
     AND  AX,NOT(CUR_WIDTH-1)
     PUSH AX
     PUSH CS:Build_Area_y      ;Push x and y of source
     PUSH CS:Build_Area_x
     CALL _BitBlt
     ADD  SP,14

     ; Clean up and return

     POP  DS                   ;Restore segment registers
     POP  ES
     POP  DI
     POP  SI

     MOV  SP,BP                ;Restore stack
     POP  BP
     RET
_Move_Cursor   ENDP

;************************************************************************
;*                                                                      *
;* _Remove_Cursor                                                       *
;*    This procedure is used to remove the cursor from the screen       *
;*    and to restore the screen to its original appearance              *
;*                                                                      *
;************************************************************************

_Remove_Cursor PROC NEAR
     PUSH BP                   ;Standard high-level entry
```

```
        MOV  BP,SP
        PUSH SI                    ;Save registers
        PUSH DI
        PUSH ES
        PUSH DS

        ; Use _BitBlt to restore area under the last cursor location

        MOV  AX,FUNC_COPY          ;Push function to use
        PUSH AX
        MOV  AX,2*CUR_HEIGHT       ;Push width and height
        PUSH AX
        MOV  AX,2*CUR_WIDTH
        PUSH AX
        PUSH CS:Last_Cursor_y      ;Push x,y of area to restore
        PUSH CS:Last_Cursor_x
        PUSH CS:Save_Area_y        ;Push x and y of save area block
        PUSH CS:Save_Area_x
        CALL _BitBlt
        ADD  SP,14

        ; Clean up and return

        POP  DS                    ;Restore segment registers
        POP  ES
        POP  DI
        POP  SI

        MOV  SP,BP                 ;Restore stack
        POP  BP
        RET
_Remove_Cursor ENDP

_TEXT   ENDS
        END
```

# Load Palette

This module is used to control the color mapping between data in display memory and the colors seen on the screen. For 16-color modes this is best done by changing the Palette registers in the Attribute controller.

As an alternative, BIOS function 10h, sub-functions 00h or 02h can be used to change color mapping.

### Listing 8-9. File: 16COL\PALETTE.ASM

```
;************************************************************************
;*                                                                      *
;* File:        PALETTE.ASM - Load palette registers                    *
;* Routine:     _Load_Palette                                           *
;* Arguments:   Start, Count, ArrayPtr                                  *
;*                                                                      *
;************************************************************************

        INCLUDE VGA.INC

        PUBLIC  _Load_Palette

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_Start       EQU     WORD PTR  [BP+4]
Arg_Count       EQU     WORD PTR  [BP+6]
Arg_ArrayPtr    EQU     DWORD PTR [BP+8]
```

```
_Load_Palette   PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Get address of input status registers

        XOR     AX,AX                   ;Segment of BIOS data area
        MOV     ES,AX
        MOV     DX,ES:[463h]            ;Fetch address of CRTC
        ADD     DX,6                    ;Compute address of input status reg

        ; Load palette registers

        IN      AL,DX                   ;Reset data/address flip/flop
        MOV     DX,3C0h                 ;Fetch address of Attribute controller
        LDS     SI,Arg_ArrayPtr         ;Fetch pointer of palette values
        MOV     AX,Arg_Start            ;Index of first palette register
        MOV     CX,Arg_Count            ;Number of registers to load

Palette_Loop:
        OUT     DX,AL                   ;Select next register
        XCHG    AH,AL                   ;Save register index
        LODSB                           ;Fetch next register value
        OUT     DX,AL                   ;Set next palette register
        XCHG    AH,AL                   ;Restore register index
        INC     AL                      ;Advance register index
        LOOP    Palette_Loop            ;Check if all done

        MOV     AL,20h                  ;Turn Attribute controller on
        OUT     DX,AL

        ; Cleanup and return

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Load_Palette   ENDP

_TEXT   ENDS
        END
```

# 9

# Programming Examples 4-Color Graphics

# Introduction

High resolution four-color graphics modes are useful for applications such as desktop publishing where resolution is important but colors are not. For the most part, these modes are from a time when only 256 K of display memory was commonly available, and color had to be sacrificed to gain higher resolution. Up to 1024 x 768 resolution can be supported with just 256 K of display memory.

Unfortunately, there is no consensus among chip manufacturers on how four-color high resolution modes should be implemented. Some VGA products even have varying levels of support between different versions of the BIOS. This chapter presents the most common organizations found on the boards we examined, and for each provides a listing for pixel read and pixel write functions. Other drawing routines can be found on the accompanying diskette.

We have seen five different memory organizations used for four-color graphics. Three of these use planar pixels and are similar to VGA mode 12h (640 x 480 16-color mode), except that different combinations of color planes are used to define pixel colors. A fifth mode uses packed pixels, and is similar to VGA mode 4 except that interleaving is not used. Each of the five types is described in the next five sections.

# Four Planes

Figure 9-1 shows the organization of display memory for four color high resolution modes with planar pixels, using all four planes of display memory. Each pixel occupies one bit position in each of two planes. Pixels in even bytes occupy planes 0 and 2, and pixels in odd bytes occupy planes 1 and 3. To convert from a pixel position, in X and Y screen coordinates, to a bit location in display memory, use the following equation:

| | |
|---|---|
| Segment | = A000h |
| Byte offset | = Video_Pitch x Y + X/8 |
| Bit position | = X modulo 8 |

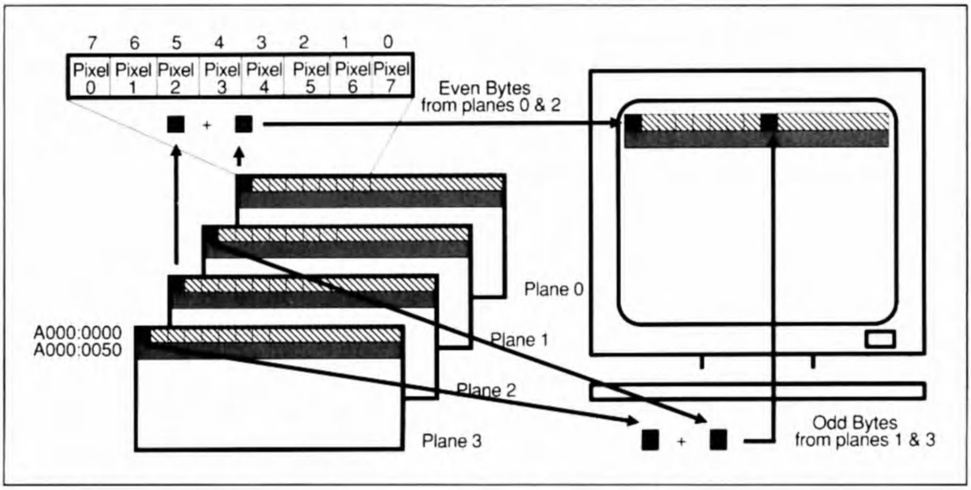


**Figure 9-1. Display memory organization - four color graphics, four planes**

The fact that some pixels are in one pair of planes and others in another, is transparent during all operations, as long as color planes are enabled properly, according to desired color. Table 9-1 contains a mapping used to map requested colors, to plane enable values. For example, a pixel of color 2 would be obtained by setting bits in planes 2 and 3 to 1, and clearing bits in planes 0 and 1 to 0.

**Table 9-1. Translating color value to plane content - four planes**

| Color | Plane 0 | Plane 1 | Plane 2 | Plane 3 |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 |
| 2 | 0 | 0 | 1 | 1 |
| 3 | 1 | 1 | 1 | 1 |

The next two programming examples show how to read and write a pixel.

# Write Pixel

Listing 9-1. File: ..\4COL\WPIXEL.ASM

```
;**********************************************************************
;*                                                                    *
;* File:        WPIXEL.ASM - 2 Bit Planar Pixel Write (alt 0&2 and 1&3) *
;* Routine:     _Write_Pixel                                          *
;* Arguments:   X, Y, Color                                           *
;*                                                                    *
;* Routine:     Select_Color                                          *
;* Arguments:   AL = Color                                            *
;*                                                                    *
;**********************************************************************

     INCLUDE VGA.INC

        EXTRN    Graf_Seg:WORD
        EXTRN    Select_Page:NEAR
        EXTRN    Video_Pitch:WORD

        PUBLIC   _Write_Pixel
        PUBLIC   Select_Color

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x            EQU     WORD PTR [BP+4]
Arg_y            EQU     WORD PTR [BP+6]
Arg_Color        EQU     BYTE PTR [BP+8]

_Write_Pixel     PROC NEAR
        PUSH     BP                     ;Preserve BP
        MOV      BP,SP                  ;Preserve stack pointer

        PUSH     ES                     ;Preserve segment and index registers
        PUSH     DS
        PUSH     DI
        PUSH     SI

        ; Convert x,y pixel addres to Page and Offset

        MOV      AX,Arg_y               ;Fetch y coordinate
        MUL      CS:Video_Pitch         ;       multiply by raster width
        MOV      CX,Arg_x               ;       add x coordinate/8
        SHR      CX,1
        SHR      CX,1
        SHR      CX,1
        ADD      AX,CX
        ADC      DX,0
        MOV      ES,CS:Graf_Seg         ;Put address in ES:DI
        MOV      DI,AX
        MOV      AL,DL                  ;Select proper page
        CALL     Select_Page

        ; Set Graphics Controller for proper color

        MOV      AL,Arg_Color           ;Fetch color to use
        CALL     Select_Color           ;Select color

        ; Set write mask

        MOV      CX,Arg_x               ;Compute X AND 7 to find mask rotation
        AND      CX,7                   ;Mask rotation is now in CL
        MOV      AL,80h                 ;Shift bit to find mask
        SHR      AL,CL                  ;Mask is now in AL
        MOV      DX,GRAPHICS_CTRL_PORT  ;Fetch graphics controller port
        MOV      AH,AL                  ;Put mask in AH
        MOV      AL,BIT_MASK_REG        ;Select bit mask register
        OUT      DX,AX                  ;Set bit mask
```

```
        ; Set pixel

        MOV      AH,ES:[DI]                ;Latch previous value
        MOV      ES:[DI],AL                ;Write color (using set/reset)

        ; Cleanup and exit

        POP      SI                        ;Restore segment and index registers
        POP      DI
        POP      DS
        POP      ES

        MOV      SP,BP                     ;Restore stack pointer
        POP      BP                        ;Restore BP
        RET
_Write_Pixel     ENDP

;**************************************************************************
;*                                                                        *
;* Routine:      Select_Color                                             *
;*               Utility routine used by all drawing routines to select   *
;*               specified color.  It is assumed that all planes are      *
;*               enabled for write, and that 'processor write' mode is    *
;*               selected.  Routine enables set/reset mechanism of VGA.   *
;* Arguments:    AL = Color                                               *
;* Returns:      DX = Points to mask select data register                 *
;*                                                                        *
;**************************************************************************

Xlat_Table        DB   00h,03h,0Ch,0FFh

Select_Color      PROC NEAR
        PUSH      AX
        PUSH      BX
        MOV       DX,GRAPHICS_CTRL_PORT    ;Use color for set/reset value
        AND       AL,03h                   ;Force color into range
        LEA       BX,CS:Xlat_Table         ;Translate color
        XLAT      CS:Xlat_Table
        MOV       AH,AL
        MOV       AL,SET_RESET_REG
        OUT       DX,AX
        MOV       DX,GRAPHICS_CTRL_PORT    ;Enable set/reset
        MOV       AL,SR_ENABLE_REG
        MOV       AH,0Fh
        OUT       DX,AX
        MOV       AL,BIT_MASK_REG          ;Select bit mask register
        OUT       DX,AL
        INC       DX
        POP       BX
        POP       AX
        RET
Select_Color      ENDP

_TEXT   ENDS
        END
```

# Read Pixel

## Listing 9-2. File: ..\4COL\RPIXEL.ASM

```
;*****************************************************************************
;*                                                                           *
;* File:       RPIXEL.ASM - 2 Bit Planar Pixel Read (read even planes)       *
;* Routine:    _Read_Pixel                                                   *
;* Arguments:  X, Y                                                          *
;* Returns:    Color in AX                                                   *
;*                                                                           *
;*****************************************************************************

        INCLUDE VGA.INC

        EXTRN     Graf_Seg:WORD
        EXTRN     Select_Page:NEAR
        EXTRN     Video_Pitch:WORD

        PUBLIC    _Read_Pixel

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x             EQU    WORD PTR [BP+4]
Arg_y             EQU    WORD PTR [BP+6]

_Read_Pixel       PROC NEAR

        PUSH      BP                      ;Preserve BP
        MOV       BP,SP                   ;Preserve stack pointer

        PUSH      ES                      ;Preserve segment and index registers
        PUSH      DS
        PUSH      DI
        PUSH      SI

        ; Convert x,y pixel addres to Page and Offset

        MOV       AX,Arg_y                ;Fetch y coordinate
        MUL       CS:Video_Pitch          ;       multiply by raster width
        MOV       CX,Arg_x                ;       add x coordinate/8
        SHR       CX,1
        SHR       CX,1
        SHR       CX,1
        ADD       AX,CX
        ADC       DX,0
        MOV       ES,CS:Graf_Seg          ;Put address in ES:DI
        MOV       DI,AX
        MOV       AL,DL                   ;Select proper page
        CALL      Select_Page

        ; Read one bit from plane 0

        MOV       DX,GRAPHICS_CTRL_PORT   ;Select Read Plane register
        MOV       AL,READ_PLANE_REG
        XOR       AH,AH                   ;Fetch plane to read (2)
        OUT       DX,AX                   ;Select plane to read

        MOV       CX,Arg_x                ;Compute X AND 7 to find mask rotation
        INC       CX
        AND       CX,7                    ;Mask rotation is now in CL
        MOV       BL,ES:[DI]              ;Get byte of video memory
        ROL       BL,CL                   ;Rotate bit into place
        AND       BL,1                    ;Mask off unneeded bits

        ; Read one bit from plane 2

        MOV       AH,2                    ;Select second plane to read
        OUT       DX,AX
```

```
        MOV     AL,ES:[DI]          ;Fetch byte from next plane
        INC     CL                  ;Update mask
        ROL     AL,CL               ;Rotate bit into place
        AND     AL,2                ;Mask off unneeded bits
        OR      AL,BL               ;Combine both bits
        XOR     AH,AH               ;Clear high order byte

        POP     SI                  ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP               ;Restore stack pointer
        POP     BP                  ;Restore BP
        RET
_Read_Pixel     ENDP

_TEXT   ENDS
        END
```

# Two Even Planes

Figure 9-2 shows a typical organization of display memory for four color high resolution modes with planar pixels, using the two even planes of display memory. Each pixel occupies one bit position in each of two planes. To convert from a pixel position, in X and Y screen coordinates, to a bit location in display memory, use the following equation:

| | |
|---|---|
| Segment | = A000h |
| Byte offset | = Video_Pitch x Y + X/8 |
| Bit position | = X modulo 8 |

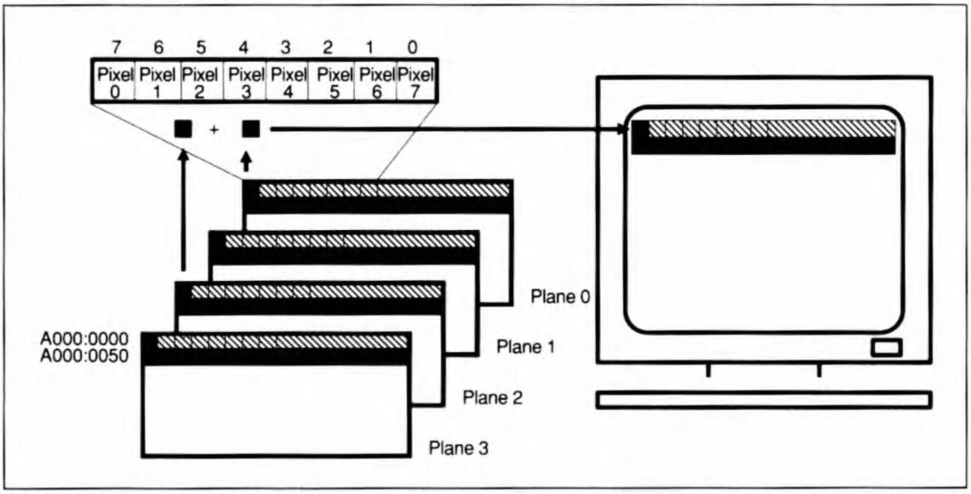


Figure 9-2. Display memory organization—4-color graphics, even planes

This memory organization is similar to the organization described in the previous section (alternating even/odd pairs), and many drawing routines are the same.

Table 9-2 shows the mapping used to map requested colors to plane enable values. For example, a pixel of color 2 would be obtained by setting bits in planes 2 to 1, and clearing bits in plane 0 to 0.

**Table 9-2. Translating color value to plane content - two even planes**

| Color | Plane 0 | Plane 2 |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 1 | 0 |
| 2 | 0 | 1 |
| 3 | 1 | 1 |

Note: Planes 1 and 3 should be disabled during drawing operations

The next two programming examples show how to read and write a pixel for this memory organization.

## Write Pixel

Listing 9-3. File: ..\4COL02\WPIXEL.ASM

```
;*********************************************************************
;*                                                                   *
;* File:        WPIXEL.ASM - 2 Bit Planar Pixel Write (even planes)  *
;* Routine:     _Write_Pixel                                         *
;* Arguments:   X, Y, Color                                          *
;*                                                                   *
;* Routine:     Select_Color                                         *
;* Arguments:   AL = Color                                           *
;*                                                                   *
;*********************************************************************

        INCLUDE VGA.INC

        EXTRN     Graf_Seg:WORD
        EXTRN     Select_Page:NEAR
        EXTRN     Video_Pitch:WORD
        PUBLIC    _Write_Pixel
        PUBLIC    Select_Color

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU     WORD PTR [BP+4]
Arg_y           EQU     WORD PTR [BP+6]
Arg_Color       EQU     BYTE PTR [BP+8]

_Write_Pixel    PROC NEAR
        PUSH    BP              ;Preserve BP
        MOV     BP,SP           ;Preserve stack pointer

        PUSH    ES              ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI
```

```
        ; Convert x,y pixel addres to Page and Offset

        MOV     AX,Arg_y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;       multiply by raster width
        MOV     CX,Arg_x                ;       add x coordinate/8
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     ES,CS:Graf_Seg          ;Put address in ES:DI
        MOV     DI,AX
        MOV     AL,DL                   ;Select proper page
        CALL    Select_Page

        ; Set Graphics Controller for proper color

        MOV     AL,Arg_Color            ;Fetch color to use
        CALL    Select_Color            ;Select color

        ; Set write mask

        MOV     CX,Arg_x                ;Compute X AND 7 to find mask rotation
        AND     CX,7                    ;Mask rotation is now in CL
        MOV     AL,80h                  ;Shift bit to find mask
        SHR     AL,CL                   ;Mask is now in AL
        MOV     DX,GRAPHICS_CTRL_PORT   ;Fetch graphics controller port
        MOV     AH,AL                   ;Put mask in AH
        MOV     AL,BIT_MASK_REG         ;Select bit mask register
        OUT     DX,AX                   ;Set bit mask

        ; Set pixel

        MOV     AH,ES:[DI]              ;Latch previous value
        MOV     ES:[DI],AL              ;Write color (using set/reset)

        ; Cleanup and exit

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Write_Pixel    ENDP

;*************************************************************************
;*                                                                       *
;* Routine:     Select_Color                                             *
;*              Utility routine used by all drawing routines to select   *
;*              specified color.  It is assumed that all planes are      *
;*              enabled for write, and that 'processor write' mode is    *
;*              selected.  Routine enables set/reset mechanism of VGA.   *
;* Arguments:   AL = Color                                               *
;* Returns:     DX = Points to mask select data register                 *
;*                                                                       *
;*************************************************************************

Xlat_Table      DB   00h,01h,04h,05h

Select_Color    PROC NEAR
        PUSH    AX
        PUSH    BX
        ; Enable only planes 0 and 2 for write
        XCHG    AX,BX                       ;Preserve color
        MOV     DX,SEQUENCER_PORT           ;Address of sequencer
        MOV     AX,PLANE_ENABLE_REG+0500h   ;Select planes 0&2 for write
        OUT     DX,AX
        XCHG    AX,BX                       ;Restore color
```

```
        ; Translate color to change planes 0 and 2
        AND     AL,03h                      ;Force color into range
        LEA     BX,CS:Xlat_Table            ;Translate color
        XLAT    CS:Xlat_Table
        MOV     AH,AL
        ; Setup Set/Reset function
        MOV     DX,GRAPHICS_CTRL_PORT       ;Use color for set/reset value
        MOV     AL,SET_RESET_REG
        OUT     DX,AX
        MOV     DX,GRAPHICS_CTRL_PORT       ;Enable set/reset
        MOV     AL,SR_ENABLE_REG
        MOV     AH,0Fh
        OUT     DX,AX
        ; Select mask register for graphics controller (used by drawing procs)
        MOV     AL,BIT_MASK_REG             ;Select bit mask register
        OUT     DX,AL
        INC     DX
        ; Clean up and return
        POP     BX
        POP     AX
        RET
Select_Color    ENDP

_TEXT   ENDS
        END
```

# Read Pixel

Listing 9-4. File: ..\4COL02\RPIXEL.ASM

```
;*************************************************************************
;*                                                                       *
;* File:        RPIXEL.ASM - 2 Bit Planar Pixel Read                     *
;* Routine:     _Read_Pixel                                              *
;* Arguments:   X, Y                                                     *
;* Returns:     Color in AX                                              *
;*                                                                       *
;*************************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pitch:WORD

        PUBLIC  _Read_Pixel

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU     WORD PTR [BP+4]
Arg_y           EQU     WORD PTR [BP+6]

_Read_Pixel     PROC NEAR
        PUSH    BP                          ;Preserve BP
        MOV     BP,SP                       ;Preserve stack pointer

        PUSH    ES                          ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Convert x,y pixel addres to Page and Offset

        MOV     AX,Arg_y                    ;Fetch y coordinate
        MUL     CS:Video_Pitch              ;       multiply by raster width
        MOV     CX,Arg_x                    ;       add x coordinate/8
        SHR     CX,1
        SHR     CX,1
```

```
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     ES,CS:Graf_Seg          ;Put address in ES:DI
        MOV     DI,AX
        MOV     AL,DL                   ;Select proper page
        CALL    Select_Page

        ; Read one bit from plane 0

        MOV     DX,GRAPHICS_CTRL_PORT   ;Select Read Plane register
        MOV     AL,READ_PLANE_REG
        XOR     AH,AH                   ;Fetch plane to read (2)
        OUT     DX,AX                   ;Select plane to read

        MOV     CX,Arg_x                ;Compute X AND 7 to find mask rotation
        INC     CX
        AND     CX,7                    ;Mask rotation is now in CL
        MOV     BL,ES:[DI]              ;Get byte of video memory
        ROL     BL,CL                   ;Rotate bit into place
        AND     BL,1                    ;Mask off unneeded bits

        ; Read one bit from plane 2

        MOV     AH,2                    ;Select second plane to read
        OUT     DX,AX
        MOV     AL,ES:[DI]              ;Fetch byte from next plane
        INC     CL                      ;Update mask
        ROL     AL,CL                   ;Rotate bit into place
        AND     AL,2                    ;Mask off unneeded bits
        OR      AL,BL                   ;Combine both bits
        XOR     AH,AH                   ;Clear high order byte

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Read_Pixel     ENDP

_TEXT   ENDS
        END
```

# Two Consecutive Planes

Figure 9-3 on page 230 shows a typical organization of display memory for four color high resolution modes with planar pixels, using planes 0 and 1 of display memory. Each pixel occupies one bit position in each of two planes. To convert from a pixel position in X and Y screen coordinates to a bit location in display memory, use the following equation:

| | |
|---|---|
| Segment | = A000h |
| Byte offset | = Video_Pitch x Y + X/8 |
| Bit position | = X modulo 8 |

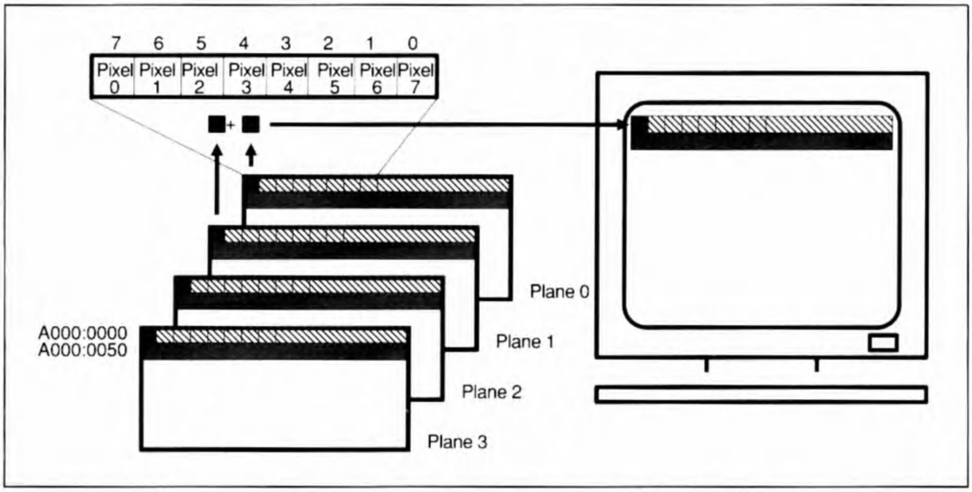


Figure 9-3. Display memory organization—4 color graphics, consecutive planes

This memory organization is similar to the organization described in the previous two sections, and many drawing routines are the same. No color mapping is needed.

The next two programming examples show how to read and write a pixel for this memory organization.

## Write Pixel

Listing 9-5. File: ..\4COL01\WPIXEL.ASM

```
;*********************************************************************
;*                                                                   *
;* File:        WPIXEL.ASM - 4 Bit Planar Pixel Write                *
;* Routine:     _Write_Pixel                                         *
;* Arguments:   X, Y, Color                                          *
;*                                                                   *
;* Routine:     Select_Color                                         *
;* Arguments:   AL = Color                                           *
;*                                                                   *
;*********************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pitch:WORD

        PUBLIC  _Write_Pixel
        PUBLIC  Select_Color

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU     WORD PTR [BP+4]
Arg_y           EQU     WORD PTR [BP+6]
Arg_Color       EQU     BYTE PTR [BP+8]
```

```
_Write_Pixel    PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Convert x,y pixel addres to Page and Offset

        MOV     AX,Arg_y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;       multiply by raster width
        MOV     CX,Arg_x                ;       add x coordinate/8
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     ES,CS:Graf_Seg          ;Put address in ES:DI
        MOV     DI,AX
        MOV     AL,DL                   ;Select proper page
        CALL    Select_Page

        ; Set Graphics Controller for proper color

        MOV     AL,Arg_Color            ;Fetch color to use
        CALL    Select_Color            ;Select color

        ; Set write mask

        MOV     CX,Arg_x                ;Compute X AND 7 to find mask rotation
        AND     CX,7                    ;Mask rotation is now in CL
        MOV     AL,80h                  ;Shift bit to find mask
        SHR     AL,CL                   ;Mask is now in AL
        MOV     DX,GRAPHICS_CTRL_PORT   ;Fetch graphics controller port
        MOV     AH,AL                   ;Put mask in AH
        MOV     AL,BIT_MASK_REG         ;Select bit mask register
        OUT     DX,AX                   ;Set bit mask

        ; Set pixel

        MOV     AH,ES:[DI]              ;Latch previous value
        MOV     ES:[DI],AL              ;Write color (using set/reset)

        ; Cleanup and exit

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Write_Pixel    ENDP

;************************************************************************
;*                                                                      *
;* Routine:     Select_Color                                            *
;*              Utility routine used by all drawing routines to select  *
;*              specified color.  It is assumed that all planes are     *
;*              enabled for write, and that 'processor write' mode is   *
;*              selected.  Routine enables set/reset mechanism of VGA.  *
;* Arguments:   AL = Color                                              *
;* Returns:     DX = Points to mask select data register                *
;*                                                                      *
;************************************************************************

Select_Color    PROC NEAR
        PUSH    AX
```

```
        MOV     DX,GRAPHICS_CTRL_PORT   ;Use color for set/reset value
        AND     AL,3                    ;Force color into range
        MOV     AH,AL
        MOV     AL,SET_RESET_REG
        OUT     DX,AX
        MOV     DX,GRAPHICS_CTRL_PORT   ;Enable set/reset
        MOV     AL,SR_ENABLE_REG
        MOV     AH,0Fh
        OUT     DX,AX
        MOV     AL,BIT_MASK_REG         ;Select bit mask register
        OUT     DX,AL
        INC     DX
        POP     AX
        RET
Select_Color    ENDP

_TEXT   ENDS
        END
```

# Read Pixel

Listing 9-6. File: ..\4COL01\RPIXEL.ASM

```
;************************************************************************
;*                                                                      *
;* File:        RPIXEL.ASM - 4 Bit Planar Pixel Read                    *
;* Routine:     _Read_Pixel                                             *
;* Arguments:   X, Y                                                    *
;* Returns:     Color in AX                                             *
;*                                                                      *
;************************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pitch:WORD

        PUBLIC  _Read_Pixel

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU     WORD PTR [BP+4]
Arg_y           EQU     WORD PTR [BP+6]

_Read_Pixel     PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Convert x,y pixel addres to Page and Offset

        MOV     AX,Arg_y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;       multiply by raster width
        MOV     CX,Arg_x                ;       add x coordinate/8
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     ES,CS:Graf_Seg          ;Put address in ES:DI
        MOV     DI,AX
        MOV     AL,DL                   ;Select proper page
        CALL    Select_Page
```

```
        ; Setup to read the value at the computed address

        MOV     DX,GRAPHICS_CTRL_PORT    ;Select Read Plane register
        MOV     AL,READ_PLANE_REG
        OUT     DX,AL
        INC     DX                       ;Point DX to data register

        MOV     AL,3                     ;Plane number
        MOV     CX,Arg_x                 ;Compute X AND 7 to find mask rotation
        AND     CX,7                     ;Mask rotation is now in CL
        MOV     BL,80h                   ;Shift bit to find mask
        SHR     BL,CL                    ;Mask is now in BL
        XOR     BH,BH                    ;Initialize return value to zero

Plane_Loop:

        OUT     DX,AL                    ;Select plane n for reading (from AL)

        ; Read byte, mask correct bit and add it into the return value

        SHL     BH,1                     ;Shift return value up
        MOV     AH,ES:[DI]               ;Get byte of video memory
        AND     AH,BL                    ;Mask out unwanted bits
        JZ      RP_Not_Set               ;Jump if bit not set
        OR      BH,1                     ;Set bit in return value
RP_Not_Set:
        DEC     AL                       ;Decrement plane number
        JGE     Plane_Loop               ;Do another plane if there are more
        MOV     AL,BH                    ;Put return value in AL
        XOR     AH,AH                    ;Clear AH

        POP     SI                       ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                    ;Restore stack pointer
        POP     BP                       ;Restore BP
        RET
_Read_Pixel     ENDP

_TEXT   ENDS
        END
```

# Four Alternating Planes

Figure 9-4 shows a typical organization of display memory for four color high resolution modes with planar pixels, using planes 0 and 1 for even pixels, and planes 2 and 3 for odd pixels. Each pixel occupies one bit position in each of two planes. Sixteen pixels are addressed by each byte. To convert from a pixel position in X,Y screen coordinates to a bit location in display memory, use the following equation:

| | |
|---|---|
| Segment | = A000h |
| Byte offset | = Video_Pitch x Y + X/16 |
| Bit position | = X modulo 16 |

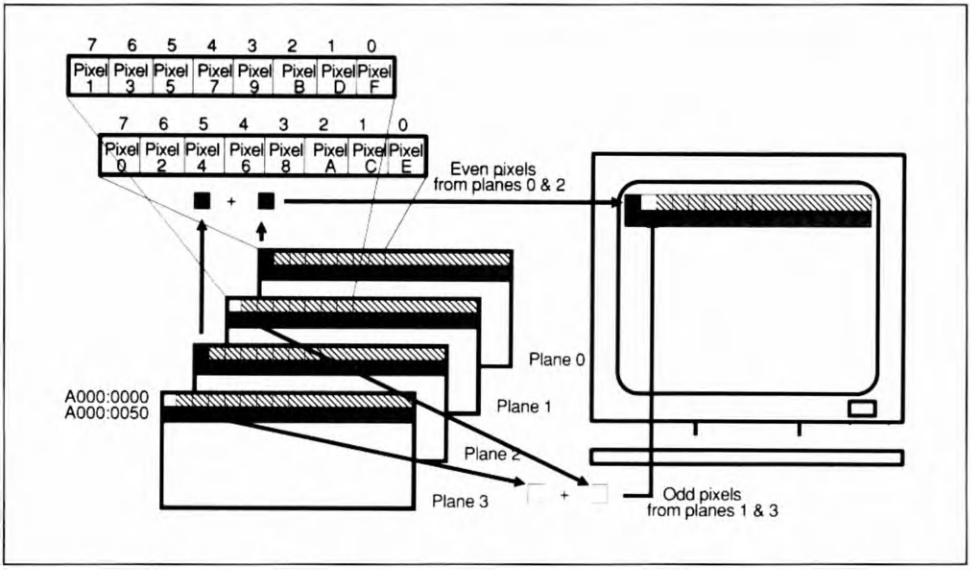


**Figure 9-4.** Display memory organization—4 color graphics, consecutive planes

This memory organization is one of the most difficult to support. Efficient drawing algorithms can be very complex. The next two programming examples show how to read and write a pixel.

## Write Pixel

Listing 9-7. File: ..\4COLATI\WPIXEL.ASM

```
;**************************************************************************
;*                                                                        *
;* File:        WPIXEL.ASM - 2 Bit Planar Pixel Write                    *
;* Routine:     _Write_Pixel                                              *
;* Arguments:   X, Y, Color                                               *
;*                                                                        *
;* Routine:     Select_Color                                              *
;* Arguments:   AL = Color                                                *
;*                                                                        *
;**************************************************************************

     INCLUDE VGA.INC

     EXTRN      Graf_Seg:WORD
     EXTRN      Video_Pitch:WORD

     PUBLIC     _Write_Pixel
     PUBLIC     Select_Color

_TEXT     SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU  WORD PTR [BP+4]
Arg_y           EQU  WORD PTR [BP+6]
```

```
Arg_Color       EQU  BYTE PTR [BP+8]

_Write_Pixel    PROC NEAR

        PUSH    BP                          ;Preserve BP
        MOV     BP,SP                       ;Preserve stack pointer

        PUSH    ES                          ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Convert x,y pixel addres to Page and Offset

        MOV     AX,Arg_y                    ;Fetch y coordinate
        MUL     CS:Video_Pitch              ;       multiply by raster width
        MOV     CX,Arg_x                    ;       add x coordinate/16
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     ES,CS:Graf_Seg              ;Put address in ES:DI
        MOV     DI,AX

        ; Set Sequencer for proper plane enable

        MOV     DX,SEQUENCER_PORT
        MOV     AL,PLANE_ENABLE_REG
        MOV     AH,03h                      ;Set plane enable for odd pixel
        TEST    Arg_x,1                     ;Check if odd pixel
        JNZ     Sel_Plane                   ;...Yes, leave enable as is
        MOV     AH,0Ch                      ;...No, set enable for even pixel
Sel_Plane:
        OUT     DX,AX                       ;Select plane

        ; Set Graphics Controller for proper color

        MOV     AL,Arg_Color                ;Fetch color to use
        CALL    Select_Color                ;Select color

        ; Set write mask

        MOV     CX,Arg_x                    ;Compute X AND 7 to find mask rotation

        SHR     CX,1
        AND     CX,7                        ;Mask rotation is now in CL
        MOV     AL,80h                      ;Shift bit to find mask
        SHR     AL,CL                       ;Mask is now in AL
        MOV     DX,GRAPHICS_CTRL_PORT       ;Fetch graphics controller port
        MOV     AH,AL                       ;Put mask in AH
        MOV     AL,BIT_MASK_REG             ;Select bit mask register
        OUT     DX,AX                       ;Set bit mask

        ; Set pixel

        MOV     AH,ES:[DI]                  ;Latch previous value
        MOV     ES:[DI],AL                  ;Write color (using set/reset)

        ; Enable all planes for write

        MOV     DX,SEQUENCER_PORT           ;Fetch sequencer port
        MOV     AX,PLANE_ENABLE_REG+0F00h   ;Set index and data
        OUT     DX,AX                       ;Enable planes

        ; Cleanup and exit

        POP     SI                          ;Restore segment and index registers
        POP     DI
        POP     DS
```

```
        POP     ES

        MOV     SP,BP                       ;Restore stack pointer
        POP     BP                          ;Restore BP
        RET
_Write_Pixel    ENDP

;*************************************************************************
;*                                                                       *
;* Routine:     Select_Color                                             *
;*              Utility routine used by all drawing routines to select   *
;*              specified color.  It is assumed that all planes are      *
;*              enabled for write, and that 'processor write' mode is    *
;*              selected.  Routine enables set/reset mechanism of VGA.   *
;* Arguments:   AL = Color                                               *
;* Returns:     DX = Points to mask select data register                 *
;*                                                                       *
;*************************************************************************

Select_Color    PROC NEAR
        PUSH    AX
        AND     AL,3                        ;Force into range
        MOV     AH,AL                       ;Duplicate color from 0-1 to 2-3
        SHL     AL,1
        SHL     AL,1
        OR      AH,AL
        MOV     DX,GRAPHICS_CTRL_PORT       ;Use color for set/reset value
        MOV     AL,SET_RESET_REG
        OUT     DX,AX
        MOV     DX,GRAPHICS_CTRL_PORT       ;Enable set/reset
        MOV     AL,SR_ENABLE_REG
        MOV     AH,0Fh
        OUT     DX,AX
        MOV     AL,BIT_MASK_REG             ;Select bit mask register
        OUT     DX,AL
        INC     DX
        POP     AX
        RET
Select_Color    ENDP

_TEXT   ENDS
        END
```

# Read Pixel

## Listing 9-8. File: ..\4COLATI\RPIXEL.ASM

```
;************************************************************************
;*                                                                      *
;* File:        RPIXEL.ASM - 2 Bit Planar Pixel Read                    *
;* Routine:     _Read_Pixel                                             *
;* Arguments:   X, Y                                                    *
;* Returns:     Color in AX                                             *
;*                                                                      *
;************************************************************************

     INCLUDE VGA.INC

     EXTRN      Graf_Seg:WORD
     EXTRN      Select_Page:NEAR
     EXTRN      Video_Pitch:WORD

     PUBLIC     _Read_Pixel

_TEXT     SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU     WORD PTR [BP+4]
Arg_y           EQU     WORD PTR [BP+6]
```

```
_Read_Pixel     PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Convert x,y pixel addres to Offset

        MOV     AX,Arg_y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;       multiply by raster width
        MOV     CX,Arg_x                ;       add x coordinate/16
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     ES,CS:Graf_Seg          ;Put address in ES:DI
        MOV     DI,AX

        ; Setup to read the value at the computed address

        MOV     DX,GRAPHICS_CTRL_PORT  ;Select Read Plane register
        MOV     AL,READ_PLANE_REG
        OUT     DX,AL
        INC     DX                      ;Point DX to data register

        MOV     AL,3                    ;Starting plane number for even pixel
        MOV     CX,Arg_x                ;Check if even pixel
        SHR     CX,1
        JC      PlaneSet                ;...Yes, leave first plane as is
        MOV     AL,1                    ;...No, set first plane for odd pixel
PlaneSet:
        AND     CX,7                    ;Compute mask
        MOV     BL,80h
        SHR     BL,CL
        XOR     BH,BH                   ;Initialize return value to zero

        ; Read byte, mask correct bit, and add it into the return value

        ;First bit
        OUT     DX,AL                   ;Select plane n for reading (from AL)
        SHL     BH,1                    ;Shift return value up
        MOV     AH,ES:[DI]              ;Get byte of video memory
        AND     AH,BL                   ;Mask out unwanted bits
        JZ      RP_Not_Set              ;Jump if bit not set
        OR      BH,1                    ;Set bit in return value
RP_Not_Set:
        ;Second bit
        DEC     AL                      ;Get next plane number
        OUT     DX,AL                   ;Select plane n for reading (from AL)
        SHL     BH,1                    ;Shift return value up
        MOV     AH,ES:[DI]              ;Get byte of video memory
        AND     AH,BL                   ;Mask out unwanted bits
        JZ      RP_Not_Set1             ;Jump if bit not set
        OR      BH,1                    ;Set bit in return value
RP_Not_Set1:

        ; Return the value just read in

        MOV     AL,BH                   ;Put return value in AL
        XOR     AH,AH                   ;Clear AH

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES
```

```
        MOV     SP,BP               ;Restore stack pointer
        POP     BP                  ;Restore BP
        RET
_Read_Pixel     ENDP

_TEXT   ENDS
        END
```

# Packed Pixels

Figure 9-5 shows a typical organization of display memory for four color high resolution modes with packed pixels. Each pixel occupies two consecutive bits in a byte. Each byte of display memory contains four pixels. The most significant two bits represent the left-most pixel for that byte. To convert from a pixel position in X,Y screen coordinates to a bit location in display memory, use the following equation:

Segment = A000h
Byte offset = Video_Pitch x Y + X/4
Bit position = X modulo 4

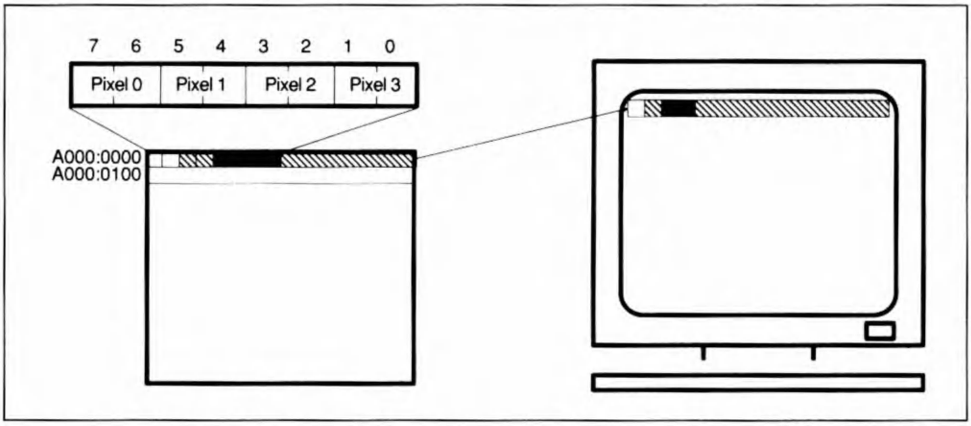


**Figure 9-5. Display memory organization—4 color graphics, packed pixels**

The next two programming examples show how to read and write a pixel for this memory organization.

# Write Pixel

Listing 9-9. File: ..\4COLPACK\WPIXEL.ASM

```
;*********************************************************************
;*                                                                   *
;* File:         WPIXEL.ASM - 2 Bit Packed Pixel Write               *
;* Routine:      _Write_Pixel                                        *
;* Arguments:    X, Y, Color                                         *
;*                                                                   *
;*********************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pitch:WORD

        PUBLIC  _Write_Pixel

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU     WORD PTR [BP+4]
Arg_y           EQU     WORD PTR [BP+6]
Arg_Color       EQU     BYTE PTR [BP+8]

_Write_Pixel    PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Convert x,y pixel addres to Page and Offset

        MOV     AX,Arg_y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;       multiply by raster width
        MOV     CX,Arg_x                ;       add x coordinate/4
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     ES,CS:Graf_Seg          ;Put address in ES:DI
        MOV     DI,AX
        MOV     AL,DL                   ;Select proper page
        CALL    Select_Page

        ; Set write mask

        MOV     CX,Arg_x                ;Compute X AND 7 to find mask rotation
        AND     CX,3                    ;Mask rotation is now in CL
        SHL     CX,1                    ;Adjust rotation for 2-bit pixels
        MOV     AL,0C0h                 ;Two-bit mask for pixel within byte
        SHR     AL,CL                   ;Rotate mask into position
        MOV     DX,GRAPHICS_CTRL_PORT   ;Fetch graphics controller port
        MOV     AH,AL                   ;Put mask in AH
        MOV     AL,BIT_MASK_REG         ;Select bit mask register
        OUT     DX,AX                   ;Set bit mask

        ; Rotate color into place

        MOV     AL,Arg_Color            ;Fetch color
        AND     AL,3                    ;Force into range
        ADD     CL,2                    ;Adjust rotation for bits in lsb
        ROR     AL,CL                   ;Rotate color into place

        ; Set pixel
```

```
        AND     BYTE PTR ES:[DI],0      ;Latch previous value and set pix to 0
        OR      ES:[DI],AL              ;Write color

        ; Cleanup and exit

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Write_Pixel    ENDP

_TEXT   ENDS
        END
```

# Read Pixel

## Listing 9-10. File: ..\4COLPACK\RPIXEL.ASM

```
;************************************************************************
;*                                                                      *
;* File:        RPIXEL.ASM - 2 Bit Packed Pixel Read                    *
;* Routine:     _Read_Pixel                                             *
;* Arguments:   X, Y                                                    *
;* Returns:     Color in AX                                             *
;*                                                                      *
;************************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pitch:WORD

        PUBLIC  _Read_Pixel

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU     WORD PTR [BP+4]
Arg_y           EQU     WORD PTR [BP+6]

_Read_Pixel     PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Convert x,y pixel addres to Page and Offset

        MOV     AX,Arg_y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;       multiply by raster width
        MOV     CX,Arg_x                ;       add x coordinate/4
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        ADC     DX,0
        MOV     DS,CS:Graf_Seg          ;Put address in DS:SI
        MOV     SI,AX
        MOV     AL,DL                   ;Select proper page
        CALL    Select_Page

        ; Compute rotation factor to move pixel color into bits 0&1
```

```
        MOV     CX,Arg_x                ;Compute X AND 3 to find mask rotation
        AND     CX,3                    ;Rotation factor is now in CL
        SHL     CX,1                    ;Adjust for 2-bit pixels
        ADD     CX,2                    ;Adjust to move pixel into bits 0&1
        AND     CX,7                    ;No need to rotate more then 8

        ; Read byte, rotate into place and mask off one pixel

        MOV     AL,[SI]                 ;Get byte of video memory
        ROL     AL,CL                   ;Rotate pixel into bits 0&1
        AND     AL,3                    ;Mask off other pixels
        XOR     AH,AH                   ;Clear AH

        ; Cleanup and return

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Read_Pixel     ENDP

_TEXT   ENDS
        END
```

# 10

# Ahead V5000
# Ahead VGA
# Wizard/Deluxe

AHEAD

# Introduction

Ahead Systems, Inc. designed the V5000 VGA chip for use on their VGA Wizard/Deluxe display adapters. At this time, two versions of the chip have been made (versions A and B). As with most SuperVGAs, the Ahead V5000 VGA chips are fully IBM VGA-compatible, include register level compatibility for EGA, CGA, MDA and Hercules, and include extended high resolution text and graphics modes. High resolution applications software drivers are also available. Ahead Systems has captured the distinction of being the first company to ship a VGA product in volume that supports 1024x768 resolution with 256 colors. Wizard/Deluxe was selected as 1990 Video Board of the Year by InfoWorld magazine.

Version B of the V5000 VGA chip contains features that are not available in version A, which is no longer being produced. Information given in this chapter applies to version B only unless stated otherwise.

# Chip Versions

Ahead V5000 VGA chips contain a version number that can be read from the least significant nibble of the Master Enable Register (I/O address 3CFh, index 0Fh). See section "Detection and Identification" for details on how the chip version can be determined.

# New Display Modes

Table 10-1 lists the enhanced display modes that are supported by the Ahead VGA Wizard/Deluxe.

# Memory Organization

For all extended display modes of the VGA Wizard/Deluxe, display memory organization is closely patterned after standard IBM VGA display modes.

For some extended modes, a memory paging mechanism is also used. Memory paging is described in detail in the programming examples.

## High Resolution Text Modes

These modes utilize memory maps that are similar to those used in standard text modes (modes 0,1,2,3, and 7), except that the number of characters per line, or number of lines per screen, is increased. Display memory is organized as shown in Figure 5-1 (see Chapter 5).

Table 10-1 Enhanced display modes—Ahead VGA Wizard/Deluxe

| Mode | Type | Resolution | Colors | Memory Required | Display Type |
|---|---|---|---|---|---|
| 22h | Text | 132 col x 44 rows | 16 | 256 KB | EGA |
| 23h | Text | 132 col x 25 rows | 16 | 256 KB | EGA |
| 24h | Text | 132 col x 28 rows | 16 | 256 KB | EGA |
| 2Fh | Text | 160 col x 50 rows | 16 | 256 KB | EGA |
| 34h | Text | 80 col x 66 rows | 16 | 256 KB | Super VGA |
| 50h | Text | 132 col x 25 rows | Mono | 256 KB | MDA |
| 52h | Text | 132 col x 44 rows | Mono | 256 KB | MDA |
| 25h,26h | Graphics | 640x480 | 16 | 256 KB | VGA |
| 60h | Graphics | 640x400 | 256 | 256 KB | VGA |
| 61h | Graphics | 640x480 | 256 | 512 KB | VGA |
| 62h | Graphics | 800x600 | 256 | 512 KB | Super VGA |
| 63h | Graphics | 1024x768 | 256 | 1024 KB | 8514 |
| 6Ah,71h | Graphics | 800x600 | 16 | 256 KB | Super VGA |
| 70h | Graphics | 720x396 | 16 | 256 KB | Super VGA |
| 74h | Graphics | 1024x768 | 16 | 512 KB | 8514 |
| 75h | Graphics | 1024x768 | 4 | 512 KB | 8514 |
| 76h | Graphics | 1024x768 | Mono | 512 KB | 8514 |

## 2-Color Graphics Mode

Memory organization for this mode resembles VGA mode 11h (640x350 2-color graphics) except that both the number of pixels per scan line and the number of scan lines are increased, and mode 76h requires paging.

## 4-Color Graphics Mode

Memory organization for this mode does not closely resemble any standard VGA modes; it somewhat resembles planar graphics mode 12h except that the memory planes are utilized differently. Planes 0 and 2 are used to store bytes at even host memory addresses. Planes 1 and 3 are used to store bytes at odd host memory addresses. See "Four Planes" in Chapter 9 to learn more about this type of memory organization.

## 16-Color Graphics Modes

Memory organization for these modes resembles VGA mode 12h (640x480 16-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Mode 74h (1024x768 16-color graphics) requires display memory

paging. Display memory organization is shown in Figure 7-1. See Chapter 7 for programming examples.

## 256-Color Graphics Modes

Memory organization for these modes resembles VGA mode 13h (320x200 256-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased, and extended modes require paging. Display memory organization is shown in Figure 8-1. See Chapter 8 for programming examples.

# New Registers

Several new registers have been added to the V5000 chip to control display memory paging and CGA/EGA/MDA emulation modes. This extended register set resides in the address space of the Graphics Controller (I/O address 3CEh/3CFh) starting at index 0Ch. Table 10-2 contains a list of the registers in the extended register set; the programming examples in this chapter contain examples showing how to access the extended registers.

**Table 10-2. Extended register set**

| Address | Index | Description | |
|---|---|---|---|
| 3CEh/3CFh | 0Ch | Mode | D7,D6 = Emulation mode |
| | | | 11 CGA |
| | | | 10 Hercules |
| | | | 01 EGA |
| | | | 00 VGA |
| | | | D5 = Enhanced mode enable |
| | | | D4 = 16 bit memory access enable |
| | | | D3 = High speed sequencer enable |
| | | | D2 = Reserved |
| | | | D1,D0 = Miscellaneous control |
| | | | 11 Reserved |
| | | | 10 Reserved |
| | | | 01 Enable 8 simultaneous fonts |
| | | | 00 Standard text mode |
| | 0Dh | Segment | D4-D7 = Write page |
| | | | D0-D3 = Read page |
| | 0Eh | Clock | D4-D7 = Divide input clock 0-3 by 2 |
| | | | D1-D3 = Reserved |
| | | | D0 = Clock 4 & 5 select enable |
| | 0Fh | Master Enable | D5 = Extended register access enable |
| | | | D0-D3 = Chip revision (READ ONLY) |

Table 10-2. **Extended register set** *(continued)*

| Address | Index | Description | |
|---|---|---|---|
| | 10h | Trap | D7 = Select 6845 as CRT controller |
| | | | D5 = Enable 3Cxh to cause traps |
| | | | D4 = Enable 3D8h, 3D9h to cause trap |
| | | | D3 = Enable 3B8h, 3BFh to cause trap |
| | | | D2 = Enable CRTC access to cause trap |
| | | | D1 = Enable 6845 access |
| | | | D0 = Enable CRTC access |
| | 11h | Trap Source | D6-D7 = Reserved |
| | | | D5 = 3Cxh |
| | | | D4 = 3BFh |
| | | | D3 = 3D9h |
| | | | D2 = 3B8h, 3D8h |
| | | | D1 = 3B5h, 3D5h |
| | | | D0 = 3Dxh |
| | 12h | Attribute | D7 = Enable CGA palette when in CGA mode |
| | | | D6 = Lock VGA internal palette |
| | | | D0-D5 = Reserved |
| | 13h | Diagnostics | D0-D7 = Reserved |
| | 14h | Lock | D7 = Lock clock select in 3C2h |
| | | | D6 = Lock CRTC index 13h |
| | | | D5 = Lock CRTC index 0Ah, 0Bh |
| | | | D4 = Lock CRTC index 9 |
| | | | D3 = Lock CRTC index 9 |
| | | | D2 = Lock CRTC vertical timing |
| | | | D1 = Lock CRTC horizontal timing |
| | | | D0 = Lock sync polarity in 3C2 |
| | 15h | 3B8h, 3D8h Readback | |
| | 16h | 3BFh, 3D9h Readback | D0-D5 = 3D9h |
| | | | D6-D7 = 3BFh bits 0 & 1 |
| | 17h | Miscellaneous | D2-D7 = Reserved |
| | | | D1 = Must be 0 |
| | | | D0 = Must be 1 |
| | 1Ch | CRTC Control | D6-D7 = Reserved |
| | | | D5 = Enable double scan |
| | | | D4 = Reserved |
| | | | D2-D3 = 00 Normal |
| | | | 01 Reserved |
| | | | 10 Reserved |
| | | | 11 Interlaced mode |
| | | | D1 = Start address bit 17 |
| | | | D0 = Start address bit 16 |
| | 1Dh | Control | D0-D7 = Reserved |

**Table 10-2. Extended register set** *(continued)*

| Address | Index | Description | |
|---|---|---|---|
| | 1Eh | Scratch | Used by BIOS for flags |
| | 1Fh | PowerUp (Read Only) | |
| | | | D4-D7 = Multiple Chip ID |
| | | | 0000 - ID 0, BIOS enabled |
| | | | 0001 - ID 1, BIOS enabled |
| | | | 0002 - ID 2, BIOS disabled |
| | | | 1111 - ID 15, BIOS disabled |
| | | | D3 = 16-bit BIOS |
| | | | D2 = 0 for 24k BIOS, 1 for 32k BIOS |
| | | | D0-D1 = Memory type |
| | | | 00 - 2 44256 DRAMs |
| | | | 01 - 4 or 16 44256 DRAMs |
| | | | 10 - 8/16 4464 DRAMs |
| | | | 11 - 8 44256 DRAMs |
| 46E8h | | Setup Control register | |
| | | | D5-D7 = Reserved |
| | | | D4 = 0 for Setup Mode, 1 for Normal Mode |
| | | | D3 = 0 for VGA disabled, 0 for VGA disabled |
| 103h | | Multiple chip ID register | |
| | | | D0-D3 = Must match Power Up register bits 0-3 |
| | | | V5000 allows up to 16 chips |
| | | | VGA Wizard/Delux allows up to 4 boards in one system |

Note: Bits marked 'reserved' must be preserved when modifying register contents.

Most registers in the extended register bank are generally not useful to the applications programmer. Listed below are the registers that we found useful enough to use in the programming examples.

## Master Enable Register (I/O Address 3CFh Index 0Fh)

D7,D6 - reserved
D5 - Extended Register Access Enable (1 = enabled)
D4 - reserved
D3-D0 - Chip Revision (read only)

**Extended Register Access Enable** must be true before any other registers in the extended register bank can be accessed.

This bit is normally set for extended graphics modes by the BIOS mode select function.

## Memory Page Select Register (I/O Address 3CFh Index 0Dh)

D7-D4 - Write page select
D3-D0 - Read page select

# Programming Examples

## Display Memory Paging

The Page Select register, located in the extended register bank at I/O address 3CFh Index 0Dh, selects which page of display memory is enabled. Two display memory pages may be selected simultaneously, one for reading and one for writing. Both pages reside at the same host memory address (normally A000:0). Dual page capability is useful when transferring data from one part of display memory to another, as for on-screen to on-screen BITBLT operations (see the BITBLT programming examples).

Figure 10-1 shows the format of the Page Select register. The read and write page may be set to the same value to achieve one memory page that is both readable and writable.

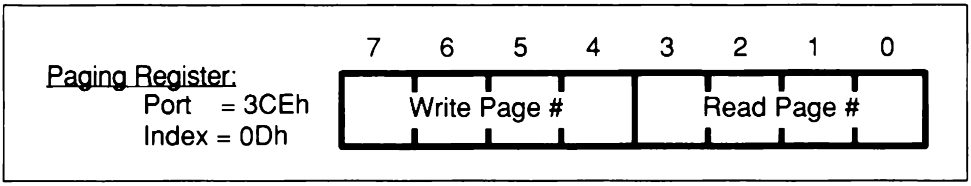


**Figure 10-1. Page Select register format—V5000 Version B**

Version A of the V5000 chip does not support dual memory pages; only one page is available. Page selection is not as straightforward as it is version B. A memory page is selected using bits 0-2 at 3CEh index 0Dh. The page can be enabled for writing using bit 5 at 3C2h and/or enabled for reading using register 3CCh. This is illustrated in Figure 10-2.

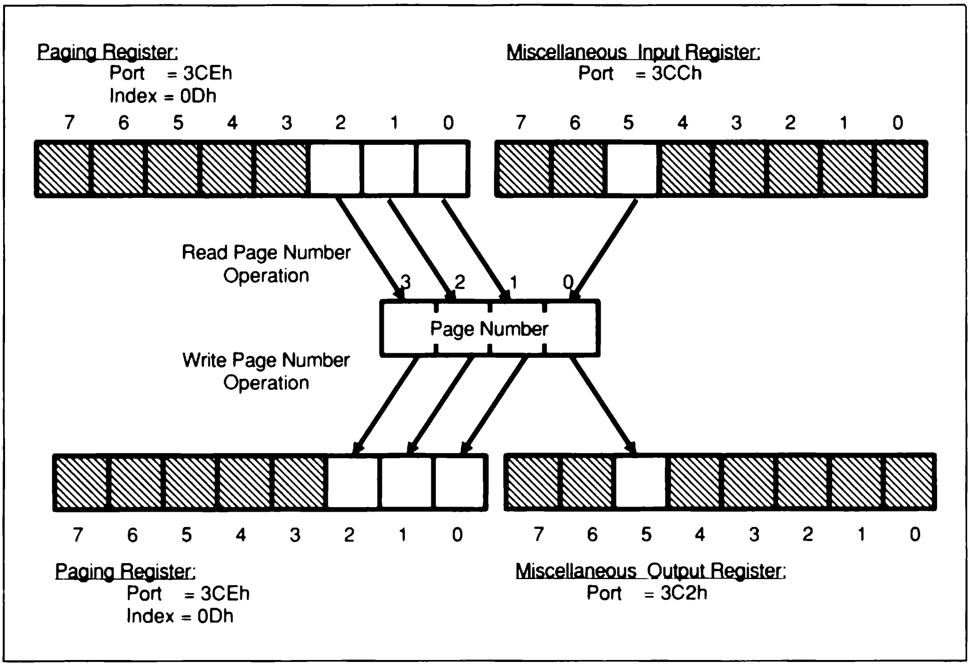


**Figure 10-2. Page Select register format—V5000 Version A**

For all graphics display modes except 256-color modes, a byte of display memory contains multiple pixels. Many drawing algorithms can be implemented efficiently by using a 'moving mask' to modify partial bytes. The BIT MASK register, index 8 of the Graphics Controller, is selected at the start of the algorithm:

```
MOV     DX,3CEh
MOV     AL,8
OUT     DX,AL
INC     DX
```

Inside the drawing loop of the algorithm, the mask data can be updated without rewriting the Index register:

```
MOV     AL,Mask
OUT     DX,AL
```

Since the Ahead Paging register resides at the same I/O address as the Graphics Controller, care must be taken to ensure that after a new page is selected, the BIT MASK register index is restored.

Display memory paging is illustrated in the following programming example. It includes a procedure _Select_Graphics to select mode (mode number is obtained from the include file MODE.INC), and three procedures for paging: _Select_Page, _Select_Read_Page and _Select_Write_Page. Note that all three page select procedures preserve the previous value of the Graphics Controller Index register to assure that drawing routines that preselect the Bit Mask register of the Graphics Controller operate properly.

**Listing 10-1. File: AHEAD\SELECT.ASM**

```
File:  AHEADSELECT.ASM
;*********************************************************************
;* File:  SELECT.ASM                                                 *
;* Description: This module contains procedures to select mode and to  *
;*        select pages.  It also initializes global variables        *
;*        according to the values in the MODE.INC include file.      *
;* Entry Points:                                                     *
;*         _Select_Graphics   - Select a graphics mode               *
;*         _Select_Text       - Set VGA adapter into text mode       *
;*         _Select_Page       - Set page for read and write          *
;*         _Select_Read_Page  - Select read page only                *
;*         _Select_Write_Page - Select write page only               *
;* Uses:                                                             *
;*        MODE.INC            - Mode dependent constants             *
;*        Following are modes and paths for Ahead boards:            *
;*   |-------- 256 colors ----------| |--16 colors --| 4 colors 2 col   *
;*   640x400 640x480 800x600 1024x768 800x600 1024x768 1024x768 1024x768*
;*Mode: 60h     61h  62h     63h     6Ah(71h)     74h    75h       76h     *
;*Path:256COL  256COL 256COL 256COL     16COL  16COL    4COL02    2COL    *
;*********************************************************************

     INCLUDE VGA.INC
     INCLUDE MODE.INC    ;Mode dependent constants

     PUBLIC    _Select_Graphics
     PUBLIC    _Select_Text
     PUBLIC    _Select_Page
     PUBLIC    _Select_Read_Page
     PUBLIC    _Select_Write_Page

     PUBLIC    Select_Page
     PUBLIC    Select_Read_Page
     PUBLIC    Select_Write_Page
     PUBLIC    Enable_Dual_Page
     PUBLIC    Disable_Dual_Page

     PUBLIC    Graf_Seg
     PUBLIC    Video_Height
     PUBLIC    Video_Width
     PUBLIC    Video_Pitch
     PUBLIC    Video_Pages
     PUBLIC    Ras_Buffer
     PUBLIC    Two_Pages

     PUBLIC    Last_Byte

;--------------------------------------------------------------------
; Data segment variables
;--------------------------------------------------------------------

;_DATA     SEGMENT WORD PUBLIC 'DATA'
;_DATA     ENDS
```

```
;-----------------------------------------------------------------------
; Constant definitions
;-----------------------------------------------------------------------


;-----------------------------------------------------------------------
; Code segment variables
;-----------------------------------------------------------------------

_TEXT     SEGMENT BYTE PUBLIC 'CODE'

Graf_Seg  DW   0A000h              ;Graphics segment addresses
          DW   0A000h
OffScreen_Seg  DW   0A000h         ;First byte beyond visible screen
Video_Pitch    DW   SCREEN_PITCH   ;Number of bytes in one raster
Video_Height   DW   SCREEN_HEIGHT  ;Number of rasters
Video_Width    DW   SCREEN_WIDTH   ;Number of pixels in a raster
Video_Pages    DW   SCREEN_PAGES   ;Number of pages in the screen
Ras_Buffer     DB   1024 DUP (0)   ;Working buffer
R_Page         DB   0FFh           ;Most recently selected page
W_Page         DB   0FFh
RW_Page        DB   0FFh
Two_Pages      DB   CAN_DO_RW      ;Indicate separate R & W capability

;***********************************************************************
;*                                                                     *
;* _Select_Graphics(HorizPtr, VertPtr, ColorsPtr)                      *
;*   Initialize VGA adapter to 640x400 mode with                       *
;*   256 colors.                                                       *
;*                                                                     *
;* Entry:                                                              *
;*   None                                                              *
;*                                                                     *
;* Returns:                                                            *
;*   VertPtr   - Vertical resolution                                   *
;*   HorizPtr  - Horizontal resolution                                 *
;*   ColorsPtr - Number of supported colors                            *
;*                                                                     *
;***********************************************************************

Arg_HorizPtr   EQU  WORD PTR [BP+4] ;Formal parameters
Arg_VertPtr    EQU  WORD PTR [BP+6] ;Formal parameters
Arg_ColorsPtr  EQU  WORD PTR [BP+8] ;Formal parameters

_Select_Graphics PROC NEAR
     PUSH BP                   ;Standard C entry point
     MOV  BP,SP

     PUSH DI                   ;Preserve segment registers
     PUSH SI
     PUSH DS
     PUSH ES

     ; Select graphics mode

     MOV  AX,GRAPHICS_MODE     ;Select graphics mode
     INT  10h

     ; Reset 'last selected page'

     MOV  AL,0FFh              ;Use 'non-existent' page number
     MOV  CS:R_Page,AL         ;Set currently selected page
     MOV  CS:W_Page,AL
     MOV  CS:RW_Page,AL

     ; Set return parameters

     MOV  SI,Arg_VertPtr       ;Fetch pointer to vertical resolution
     MOV  WORD PTR [SI],SCREEN_HEIGHT   ;Set vertical resolution
     MOV  SI,Arg_HorizPtr      ;Fetch pointer to horizontal resolution
     MOV  WORD PTR [SI],SCREEN_WIDTH    ;Set horizontal resolution
```

```
     MOV  SI,Arg_ColorsPtr    ;Fetch pointer to number of colors
     MOV  WORD PTR [SI],SCREEN_COLORS   ;Set number of colors

     ; Clean up and return to caller

     POP  ES                  ;Restore segment registers
     POP  DS
     POP  SI
     POP  DI

     MOV  SP,BP               ;Standard C exit point
     POP  BP
     RET
_Select_Graphics ENDP

;*************************************************************************
;                                                                        *
; Select_Page                                                            *
; Entry:                                                                 *
;    AL - Page number                                                    *
;                                                                        *
;*************************************************************************

Select_Page    PROC NEAR
     CMP  AL,CS:RW_Page       ;Check if already selected
     JNE  SP_Go
     RET
SP_Go:
     PUSH AX
     PUSH BX
     PUSH DX
     ;Save currently selected page number
     AND  AL,0Fh              ;Force page number into range
     MOV  CS:RW_Page,AL       ;Save as most recent RW page
     MOV  CS:R_Page,AL        ;Invalidate R and W pages
     MOV  CS:W_Page,AL
     ;Fetch gr. ctrl. index (some drawing routines need it preserved)
     MOV  DX,3CEh             ;Fetch address of page select
     XCHG BL,AL               ;Save AL
     IN   AL,DX               ;Must save current gr. ctrl. index
     XCHG BL,AL
     ;Move page number into proper bits
     MOV  AH,AL               ;Copy page number into high nibble
     SHL  AL,1
     SHL  AL,1
     SHL  AL,1
     SHL  AL,1
     OR   AH,AL
     ;Select new page
     MOV  AL,0Dh              ;Fetch page register index
     OUT  DX,AX               ;Write out the new page select
     ;Restore gr. ctrl. index
     XCHG AL,BL               ;Restore gr. ctrl. index
     OUT  DX,AL

     POP  DX
     POP  BX
     POP  AX
     RET
Select_Page    ENDP

;*************************************************************************
;                                                                        *
; Select_Read_Page                                                       *
; Entry:                                                                 *
;    AL - Page number                                                    *
;                                                                        *
;*************************************************************************

Select_Read_Page PROC NEAR
     CMP  AL,CS:R_Page        ;Check if already selected
```

```
     JNE  SRP_Go
     RET
SRP_Go:
     PUSH AX
     PUSH BX
     PUSH DX
     ; Save new values
     MOV  CS:RW_Page,0FFh     ;Invalidate RW page value
     AND  AL,0Fh              ;Force page # into range
     MOV  CS:R_Page,AL
     MOV  AH,AL               ;Save page number
     ;Fetch gr. ctrl. index (some drawing routines need it preserved)
     MOV  DX,3CEh             ;Fetch address of page select
     IN   AL,DX               ;Must save current gr. ctrl. index
     MOV  BL,AL
     ;Move page number into proper bits and select new page
     MOV  AL,0Dh              ;Fetch page register index
     OUT  DX,AL               ;Select register
     INC  DX
     IN   AL,DX               ;Fetch previous value of page reg
     AND  AL,0F0h             ;Preserv write page
     OR   AL,AH               ;Move page number into ""read" bits
     OUT  DX,AL               ;Write out the new page select
     ;Restore graphics controller index
     MOV  AL,BL               ;Restore gr. ctrl. index
     DEC  DX
     OUT  DX,AL
     ; Clean up and return
     POP  DX
     POP  BX
     POP  AX
     RET
Select_Read_Page ENDP

;*************************************************************************
;                                                                        *
; Select_Write_Page                                                      *
; Entry:                                                                 *
;    AL - Page number                                                    *
;                                                                        *
;*************************************************************************

Select_Write_Page PROC NEAR
     CMP  AL,CS:W_Page        ;Check if already selected
     JNE  SWP_Go
     RET
SWP_Go:
     PUSH AX
     PUSH BX
     PUSH DX
     ; Save new values
     MOV  CS:RW_Page,0FFh     ;Invalidate RW page value
     MOV  CS:W_Page,AL        ;Save new write value
     MOV  AH,AL
     ;Fetch gr. ctrl. index (some drawing routines need it preserved)
     MOV  DX,3CEh             ;Fetch address of page select
     IN   AL,DX               ;Must save current gr. ctrl. index
     MOV  BL,AL
     ;Move page number into proper bits and select new page
     SHL  AH,1                ;Copy page # into hi nibble of AH
     SHL  AH,1
     SHL  AH,1
     SHL  AH,1
     MOV  AL,0Dh              ;Fetch page register index
     OUT  DX,AL               ;Select register
     INC  DX
     IN   AL,DX               ;Get current values
     AND  AL,0Fh              ;Preserve read page number
     OR   AL,AH               ;Move page number into ""write" bits
     OUT  DX,AL               ;Write out the new page select
     ;Restore graphics controller index
```

```
     MOV  AL,BL              ;Restore gr. ctrl. index
     DEC  DX
     OUT  DX,AL
     ; Clean up and return
     POP  DX
     POP  BX
     POP  AX
     RET
Select_Write_Page ENDP

;*************************************************************************
;*                                                                       *
;* Enable_Dual_Page                                                      *
;* Disable_Dual_Page                                                     *
;*   Not supported by Ahead based boards                                 *
;*                                                                       *
;*************************************************************************

Enable_Dual_Page    PROC NEAR
     RET
Enable_Dual_Page    ENDP

Disable_Dual_Page   PROC NEAR
     RET
Disable_Dual_Page   ENDP

;*************************************************************************
;                                                                        *
; _Select_Page(PageNumber)                                               *
; Entry:                                                                 *
;    PageNumber - Page number                                            *
;                                                                        *
;*************************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Page   PROC NEAR
     PUSH BP                 ;Setup frame pointer
     MOV  SP,BP
     MOV  AL,Arg_PageNumber  ;Fetch argument
     POP  BP                 ;Restore BP
     JMP  Select_Page
_Select_Page   ENDP

;*************************************************************************
;                                                                        *
; _Select_Read_Page(PageNumber)                                          *
; Entry:                                                                 *
;    PageNumber- Page number for read                                    *
;                                                                        *
;*************************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Read_Page   PROC NEAR
     PUSH BP                 ;Setup frame pointer
     MOV  SP,BP
     MOV  AL,Arg_PageNumber  ;Fetch argument
     POP  BP                 ;Restore BP
     JMP  Select_Read_Page
_Select_Read_Page   ENDP

;*************************************************************************
;                                                                        *
; _Select_Write_Page(PageNumber)                                         *
; Entry:                                                                 *
;    PageNumber - Page number for write                                  *
;                                                                        *
;*************************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]
```

```
_Select_Write_Page  PROC NEAR
     PUSH BP                  ;Setup frame pointer
     MOV  SP,BP
     MOV  AL,Arg_PageNumber   ;Fetch argument
     POP  BP                  ;Restore BP
     JMP  Select_Write_Page
_Select_Write_Page  ENDP

;*********************************************************************
;*                                                                   *
;* _Select_Text                                                      *
;*   Set VGA adapter to text mode                                    *
;*                                                                   *
;*********************************************************************

_Select_Text   PROC NEAR
     MOV       AX,TEXT_MODE        ;Select mode 3
     INT       10h                 ;Use BIOS to reset mode
     RET
_Select_Text   ENDP

Last_Byte:
_Text     ENDS
          END
```

# Detection and Identification

Ahead VGA cards can be detected by a signature field located in the Ahead BIOS ROMs at location C000:0025h, containing the ASCII characters 'AHEAD'. Chip version can be obtained from register index 20h in the extended register set. Version A chips return a value of 20h, and version B chips return a value of 21h. For example:

```
          MOV          DX,3CEh              ;Fetch I/O Address
          MOV          AL,0Fh               ;Fetch index of 'Enable' reg
          OUT          DX,AL                ;Select 'Master Enable' register
          INC          DX                   ;I/O address of data
          MOV          AL,20h               ;Fetch ENABLE value
          OUT          DX,AL                ;Enable extended register set
          JMP          $+2                  ;Wait for I/O to complete
          IN           AL,DX                ;Fetch chip version
          TEST         AL,1                 ;Test for version B
          JNZ          VersionB
VersionA:

VersionB:
```

# 11

# ATI 18800
# ATI VGAWONDER

TECHNOLOGIES INC.
Technology you can Trust.

# Introduction

By developing their own VLSI VGA controller chip, ATI achieved true BIOS and register compatibility not only with VGA, but with the EGA, CGA, MDA, and Hercules display adapters as well. In addition to VGA-compatible analog displays, the VGA *WONDER* can also drive the TTL displays that are compatible with other video adapters.

The VGA*WONDER* can be purchased in either of two memory configurations: 256K of DRAM or 512K of DRAM. Some of the enhanced display modes of the adapter can only be used if a full 512K of DRAM is present.

# Versions of the Adapter

At the heart of the VGA*WONDER* is the ATI18800 controller chip, a VLSI integrated circuit developed by ATI Technologies. Two versions of this device have been used on the VGA*WONDER*. They will be referred to here as the Rev.1 chip and the Rev.2 chip.

Another device, the ATI18810 Video Dot Clock Generator, was developed by ATI to generate the many different clock frequencies required to support multiple resolutions. This device replaces several oscillator devices that would otherwise be required.

This chapter applies to three different versions of the VGA*WONDER* adapter. Table 11-1 lists the three adapter versions and shows which chip versions each of the adapters uses. To determine the version of our board, see the BIOS ROM constants described in the section "Identifying the VGA*WONDER*."

**Table 11-1.** VGA*WONDER* versions

| Board Version | V3 | V4 | V5 |
|---|---|---|---|
| ATI18800 version | Rev.1 | Rev.2 | Rev.2 |
| ATI18810 used | No | No | Yes |
| ROM BIOS label | V3M-x.xx | V4M-x.xx | V5M-x.xx |

Differences between the Rev.1 and Rev.2 controller chip will be noted in detail later in this chapter.

# New Display Modes

Table 11-2 lists the enhanced display modes that are supported by VGA*WONDER*.

Table 11-2. Enhanced display modes — VGA*WONDER*

| Mode | Type | Resolution | Colors | Memory Required | Display Type |
|---|---|---|---|---|---|
| 23h | Text | 132 col x 25 rows | 16 | 256 KB | EGA |
| 27h | Text | 132 col x 25 rows | mono | 256 KB | EGA |
| 33h | Text | 132 col x 44 rows | 16 | 256 KB | EGA |
| 37h | Text | 132 col x 44 rows | mono | 256 KB | EGA |
| 54h,6Ah | Graphics | 800x600 | 16 | 256 KB | SuperVGA |
| 55h (1) | Graphics | 1024x768 | 16 | 512 KB | 8514 or XL |
| 61h | Graphics | 640x400 | 256 | 256 KB | VGA |
| 62h | Graphics | 640x480 | 256 | 512 KB | VGA |
| 63h | Graphics | 800x600 | 256 | 512 KB | SuperVGA |
| 65h | Graphics | 1024x768 | 16 | 512 KB | 8514 or XL |
| 67h | Graphics | 1024x768 | 4 | 256 KB | 8514 or XL |

NOTE: Boards with chip version 1 do not support mode 55h.

It is important to verify that the display being used is capable of supporting the colors and resolution of the selected display mode, and that the VGA*WONDER* has been properly configured for that display type. Otherwise, the BIOS mode-select function may not initialize the mode properly.

A utility program VCONFIG, normally supplied with the VGA*WONDER*, can be used to configure the board.

# Memory Organization

For most extended modes, display memory organization is patterned after the organization used in one or more standard IBM VGA modes. For some modes, the memory organization is totally different from any previous industry precedents.

VGA*WONDER* includes a display memory paging mechanism that is needed in some display modes to make the entire display memory accessible to the processor. Display memory paging is described in detail later in this chapter.

## High Resolution Text Modes (23h, 27h, 33h, 37h)

These modes utilize memory maps that are similar to those used in standard text modes (modes 0, 1, 2, 3 and 7), except that the number of characters per line is increased from 80 to 132. This increases the number of bytes used per text line from 160 to 264. Display memory is organized as shown in Figure 5-1 (see Chapter 5).

## High Resolution Graphics Modes

### *Modes 54h - 800x600 (16 colors)*

Memory organization for this mode resembles VGA mode 12h (640x480 16-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is shown in Figure 7-1. See Chapter 7 for programming examples.

Only 256K of display memory are required to support this mode; display memory paging is not required.

### *Mode 55h - 1024x768 (16 colors)*

Memory organization for this mode resembles VGA mode 12h (640x480 16-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is shown in Figure 7-1. See Chapter 7 for programming examples.

To support this mode, 512K of display memory are required. Display memory paging is required. Default colors are the same as for mode 12h (16-color graphics).

### *Modes 61h, 62h, 63h (256 colors)*

These modes, because of their higher resolutions, require larger amounts of display memory which exceed the 64K page size of display memory. The Memory Page Select register, in the extended register bank, is used to select which memory page can be accessed by the processor.

Display memory organization for these modes resembles VGA mode 13h (320x200 256-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. The memory map for these modes can be seen in Figure 8-1 (see Chapter 8). Default colors are the same as for mode 13h.

### *Mode 65h - 1024x768 (16-colors)*

Display memory organization in this mode does not resemble any standard VGA mode. Instead of the planar pixels used in other 16-color modes, packed pixels are used. Pixels are packed two per byte. Color planes are not used; the memory is mapped as a single memory plane of 512K, which is segmented into eight pages of 64K each. Memory pages are selected using the Page Select register in the extended register bank. To display one screen at 1024x768 resolution 384K of display memory are required.

Because this mode is unique to ATI, a separate set of programming examples is included in the ATI directory. Listings for the Write Pixel and Read Pixel routines are shown in the text at the end of this chapter. See Figure 11-1 for the memory map used in mode 65h.

Default colors are the same as for mode 12h (16-color graphics), but the palette registers are programmed differently. If the color palette is to be modified in this mode, it must be done by modifying the DAC registers. The palette registers of the Attribute Controller should not be altered while in this mode. The first sixteen DAC registers, registers 00h through 0Fh should be loaded with the desired colors, and then the fifteen registers at 10h, 20h, 30h, ..., F0h should be set to match registers 0 through 0Fh (register 10h should match register 01h, 20h should match 02h, 30h should match 03h, etc.).

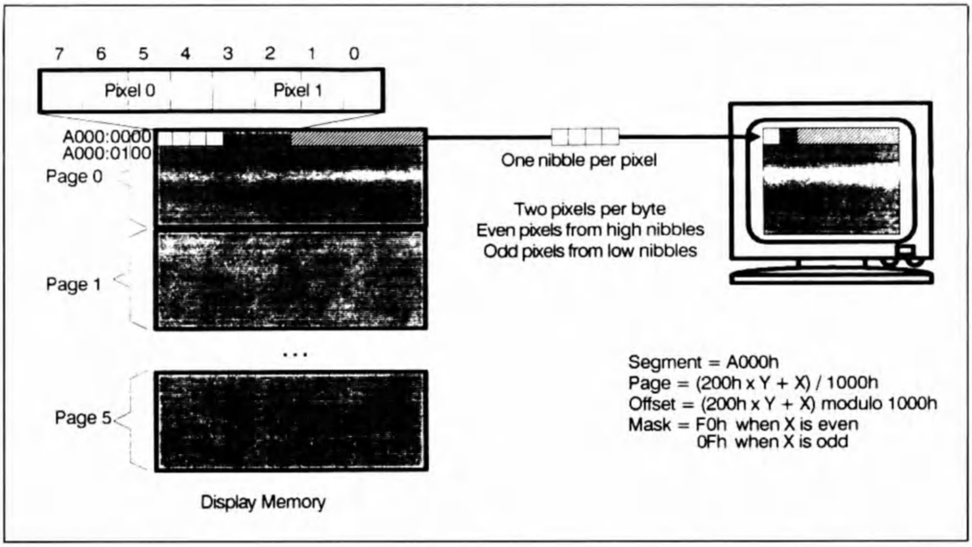


**Figure 11-1. Display memory organization — mode 65h**

## *Mode 67h - 1024x768 (4 Colors)*

This mode is also unique for ATI cards. Display memory is organized as two sets of two memory planes. Each pixel requires two bits of display memory; planes 0 and 1 contain odd numbered pixels and planes 2 and 3 contain even numbered pixels. A single host memory byte address addresses sixteen pixels (see Figure 11-2 on the following page). Only 256K of display memory are required for this mode. Memory paging is not required.

Four standard color sets are supported in this mode, as shown in Table 11-3. Colors are selected using bits 0 and 1 of the Color Select register of the Attribute Controller (port 3C0h, index 34h).

**Table 11-3 Mode 67h color sets**

| Color Set | Color Index and Result | | | |
|---|---|---|---|---|
| D1 D0 | 00 | 01 | 10 | 11 |
| 0 0 | Black | White | Gray | Intens. White |
| 0 1 | Black | Cyan | Red | White |
| 1 0 | Black | Green | Red | Yellow |
| 1 1 | Black | Cyan | Magenta | White |

Note: The Mode Control register of the Attribute Controller (index 10h) must have bit 7 set to one to enable operation of the Color Select register.

To learn more about mode 67h see section "Four Alternating Planes" in Chapter 9.

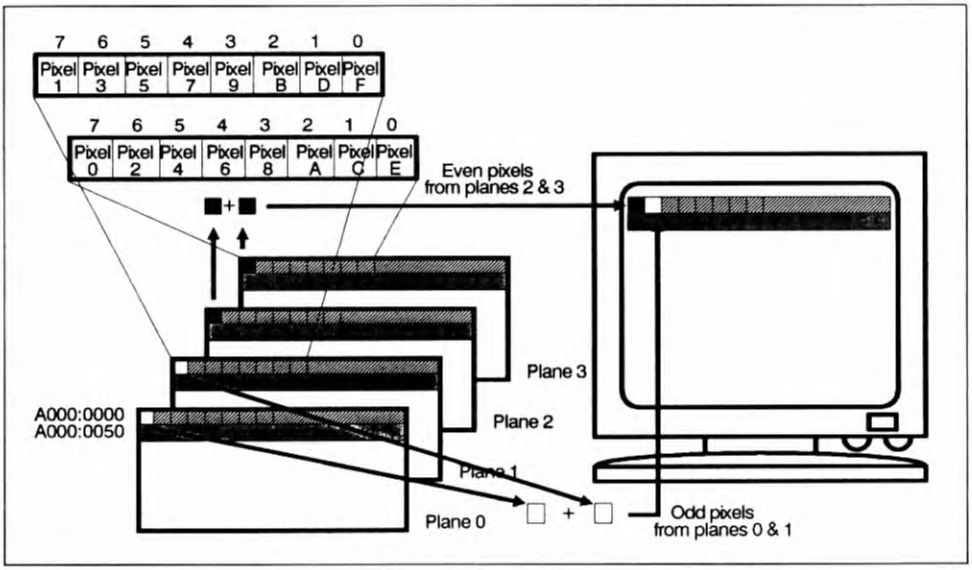


Figure 11-2. Display memory organization — mode 67h

# New Registers

Contained within the VGA controller chip on the VGA*WONDER* is an extended bank of registers that is used to access the advanced features of the adapter. All registers in the extended register bank have read and write capability.

The default I/O address of the extended register bank is stored in the BIOS ROM at memory address C000:10h. To guarantee compatibility with future ATI products, software written for the VGA*WONDER* should not assume that this I/O address will remain constant.

The extended register bank must be treated slightly differently than other VGA registers. After an index value is written to the index register, the data register may only be written or read once; the index must be rewritten before another access is made. All input from the extended register bank must be performed in bytes; the instruction **IN AX,DX** will not work properly. To write extended registers, ALWAYS use the word output instruction **OUT DX,AX**.

Care should be taken not to modify registers which are not described. All register bits marked as reserved should be preserved if the register is modified. The following code can be used to access a particular register in the extended register bank.

Programming examples that illustrate the proper methods for reading and writing extended registers can be found later in this chapter under "Programming Examples." The following code can be used to access a particular register in the extended register bank:

```
; Fetch I/O address of ATI extended registers
MOV          AX,0C000h               ;Fetch segment address of VGA BIOS
MOV          DS,AX
MOV          SI,10h                  ;Fetch offset of ATI register address
MOV          DX,DS:[SI]              ;Get I/O address of ATI registers

; Fetch value of extended register XXXX (Input)

MOV          AL,XXXX                 ;Get index of desired register
OUT          DX,AL                   ;Select register index
INC          DX                      ;Advance port number to data register
IN           AL,DX                   ;Read the register data
DEC          DX                      ;Restore port number

; Write new value to extended register XXXX (Output)

MOV          AL,XXXX                 ;Get index of desired register
MOV          AH,Data                 ;Get data value
OUT          DX,AX                   ;MUST WRITE IT AS A WORD
```

Table 11-4 on page 264 shows the registers of the extended register bank.

Table 11-4. Extended register bank — VGA*WONDER*

| Index | Register |
|---|---|
| B0h | DRAM timing |
| B1h | EGA compatibility and double scanning enable |
| B2h | Memory page select register |
| B3h | Enable 1024 x 768 graphics |
| B4h | Emulation control |
| B5h | Misc. control |
| B6h | High resolution enable |
| B7h | Reserved |
| B8h | Register write protect and clock select |
| B9h | Miscellaneous control |
| BAh | Miscellaneous control |
| BBh | Input status register |
| BCh | EGA switch settings |
| BDh | Miscellaneous control |
| BEh | Miscellaneous register |

## ATI Register 0 (Index B0h)

D7 - Reserved
D6 - Hercules 300 line emulation
D1, D2, D4, D5 - DRAM timing
D3 - Enable 8 CRT accesses for each CPU access
D0 - Reserved

## ATI Register 1 - EGA Compatibility and Double Scanning Enable (Index B1h)

D7 - Reserved
D6 - Divide vertical timing parameters by 2 (1 = true)
D5-D3 - double scanning / 3 of 4 scanning enable
    001 - Enable double scanning in graphics mode
    010 - Enable 3 of 4 scanning in graphics mode
    101 - Enable double scanning in text mode
    110 - Enable 3 of 4 scanning in text mode
D2 - General purpose read/write
D1 - Force all registers to be EGA compatible (1 = true)
D0 - Force all I/O addresses to be EGA compatible (1 = true)

## ATI Register 2 - Memory Page Select (Index B2h)

The Revision 2 VGA chip has additional paging capabilities that the Revision 1 chip does not.

For the Revision 1 chip:

D7 - Reserved
D6 - External clock select
D5 - Enable internal DIP switch settings (EGA mode)
D4-D1 - Display memory page select
D0 - Enable interlace mode (1 = true)

For the Revision 2 chip:

D7-D5 - Read Page select
D4 - Reserved
D3-D1 - Page select
D0 - Reserved

When operating the VGA*WONDER* in high resolution graphics modes that require more than one page (64K) of memory per plane, the Memory Page Select register is used to select which 64K page can be accessed by the host CPU.

For the Revision 1 ATI VGA chip, only one memory page may be selected at a time. The Revision 2 chip has an optional mode that allows two pages to be selected simultaneously, one page being read only and one page being write only. For compatibility, the Revision 2 chip defaults on initialization to the single page mode.

Dual page mode is enabled through the Miscellaneous register in the extended register bank (index BEh).

To determine which revision chip is present on an adapter, see "Detection and Identification" in the "Programming Examples" section.

## ATI Register 3 (Index B3h)

For the Revision 1 chip, control of this register should be left to the VGA BIOS. This register should not be modified.

D7 - Reserved
D6 - Reserved
D5 - Enable 16-bit operation
D4 - Enable PS/2 decoding
D3 - EEPROM chip select
D2 - Enable EEPROM interface

D1 - EEPROM clock source
D0 - EEPROM data input

For the Revision 2 chip:

D7 - Enable double scanning for 200-line modes (Rev. 2. only)
D6 - Enable 1024x768 16-color planar pixel mode (Rev. 2 only)
D5 - Enable 16-bit operation
D4 - Disable memory beyond 256K
D3 - EEPORM chip select
D2 - Enable EEPROM interface
D1 - EEPROM clock source
D0 - EEPROM data input

## ATI Register 4 (Index B4h)

D7 - Override locking of CR117
D6 - Lock CR0-CR7 instead of CR117
D5 - Lock CR80-CR86 and CR140-CR144
D4 - Lock cursor start and end
D3 - Lock vertical timing registers
D2 - lock CR90-94, CR97
D1 - Enable Hercules emulation
D0 - Enable CGA emulation

## ATI Register 5 (Index B5h)

D7 - reserved
D6 - Enable CGA Cursor Emulation
D5 - Disable Cursor Blinking (1 = disabled)
D4 - Enable 8 simultaneous fonts
D3 - Select Map 3 as programmable character generator
D2 - Enable display signal skew
D1 - Invert blanking signal polarity
D0 - Select display enable as blanking signal

Enable CGA Cursor Emulation, when set to 1, adds five to the cursor start and end registers, so that a cursor which is set by software to work in the CGA 8x8 character cell will appear properly in a larger 8x14 character cell.

Disable Cursor Blinking forces the cursor to display steadily without blinking.

## ATI Register 6 (Index B6h)

D7 - Disable blanking screen blank in CGA and Hercules emulation
D6 - Select composite sync for output
D5 - Enable vertical interrupt
D4 - Select 16-color high res modes
D3 - Select 4-color high res modes
D2 - Reserved
D1 - Enable 640x400 Hercules emulation
D0 - Reserved

## ATI Register 7 (Index B7h)

D0 to D7 - Reserved

## ATI Register 8 (Index B8h)

D7, D6 - Clock divider
D5 - Lock vertical sync polarity
D4 - Lock horizontal sync polarity
D3 - Lock write to 3C2h
D2 - Lock all VGA registers except CRTC start and end
D1 - Lock Overscan register in Attribute Controller
D0 - Lock Palette registers in Attribute Controller

## ATI Register 9 (Index B9h)

D7 - Lock Line Compare register
D6 - Set horizontal total = register value + 2 (vs +5)
D4, D5 - Wait cycles for 16 bit access to ROM
D3, D2 - ROM address space
D1 - Select input to clock chip
D0 - Clock select

## ATI Register A (Index BAh)

D7 - Delay chain resolution compensation
D6 - Reserved
D5 - Enable monochrome gray scale circuit
D4 - Enable EGA color simulation for RGB monitors
D3 - Disable secondary red output (for RGB monitors)
D2-D0 - Delay chain timing compensation

## ATI Register B - Input Status Register (Index BBh)

This register is actually just a one byte read/write latch which is set up by the VGA*WONDER* BIOS to contain the following information:

D7 - Reserved
D6 - Reserved
D5 - Memory size (0 = 256K, 1 = 512K)
D4 - Reserved
D3-D0 - Display Type. The board is configured for:
0 = EGA
1 = PS/2 Analog monochrome
2 = TTL monochrome
3 = PS/2 color
4 = Analog RGB
5 = Multisync or similar
7 = IBM 8514
9 = NEC VGA monitor
D = NEC Multisync XL

## ATI Register C (Index BCh)

D0 to D7 - Reserved

## ATI Register D (Index BDh)

D4 to D7 - EGA switch settings
D0 to D3 - Reserved

## ATI Register E - Miscellaneous Register (Index BEh - Rev. 2 only)

This register is only present in the Revision 2 ATI VGA chip.

D7 - enable 1024x768 4-color mode
D6 - enable 1024x768 16-color mode
D5,D4 - reserved
D3 - Enable dual page Mode
D2 - Select internal EGA DIP Switch value
D1 - Enable interlaced mode
D0 - Unlock Vertical Display End register of the CRT Controller

For an explanation of dual page mode, see the Page Select register (index B2h).

# The BIOS

All modes of the VGA*WONDER* can be set using the BIOS Mode Set command (function 0). In addition, the VGA*WONDER* BIOS supports a new command that will return a pointer to the BIOS parameter table (the table that is used to initialize registers during a mode set) so that registers can be loaded directly. Extended text modes are fully supported by all functions of the ROM BIOS. In enhanced graphics modes, however, only the Mode Set and Load Palette BIOS functions are supported.

The following sequence can be used to invoke an extended display mode:

```
MOV        AH,0                  ;Setup mode select function
MOV        AL,MODE_NUMBER        ;Setup mode number
INT        10H                   ;Select mode by using BIOS
```

## Extended BIOS Functions

### *BIOS function 12h Sub function 6 - Get Parameter Table Pointer*

**Input Parameters:**

AH = 12h
BL = 6
BH = 55h
AL = Mode Number
BP = 0FFFFh (set to known "illegal" value)
SI = 0 (set to know "illegal" value)

**Return Value:**

ES:BP = Pointer to parameter table
BP = Remains unchanged if the requested mode is not supported in this configuration
ES:SI = pointer to table override pairs (table index, table value) terminated with index 3Fh

Example:

```
MOV      AH,12h            ;Select BIOS function
MOV      BX,5506h          ;Sub-function 6
MOV      AL,Mode_Number    ;Desired display mode number
MOV      BP,0FFFFh         ;Initialize BP to known invalid value
XOR      SI,SI
INT      10h               ;Do BIOS call
CMP      BP,0FFFFh         ;Check for error
JE       Bad_Mode
```

After a successful return from the Get Parameter Table, the ES:BP points to a parameter table that is formatted the same as the parameter table that is included in the BIOS Environment Table. ES:BP points to the extended register values for that mode. If any values in the table need modification (e.g., due to special configuration), then ES:BP points to a list of pairs that define the override values. The first byte in the pair contains an index into the parameters table returned in ES:BP and the second byte contains the replacement value. The table is terminated with index 3Fh.

## Extended BIOS Data Area

The VGA BIOS is located in the system memory space starting at address C000:0000. At the beginning of the BIOS are several constants which are used to determine the version and capabilities of the adapter. These constants are listed in Table 11.5.

**Table 11-5. VGA*WONDER* BIOS constants**

| ROM Address | ROM Data | Description |
|---|---|---|
| C000:10 | WORD | I/O address of Extended Register Block (see Note 1) |
| C000:31 | "761295520" | ASCII ATI signature found in all ATI BIOS products |
| C000:40 | "31" | ASCII VGA*WONDER* signature code |
| | "32" | ASCII EGA*WONDER* 800 + signature code |
| C000:42 | BYTE | D0 = 1: Can switch between 8 or 16 bit ROM |
| | | D1 = 1: mouse interface on board |
| | | D4 = 1: Use clock chip |
| | | D7 = 1: Use C000:0000 to D000:FFFF with 16-bit ROM |
| C000:43 | '1' | VGA*WONDER* with version 1 chip |
| | '2' | VGA*WONDER* with version 2 chip (see Note 2) |
| | '3' | VGA*WONDER* with version 2 chip, VRAM version |
| C000:4C | BYTE | Major BIOS revision number (binary) |
| C000:4D | BYTE | Minor BIOS revision number (binary) |

Note 1: If address C000:0010 cannot be accessed dynamically, use 1CEh as the I/O address.
Note 2: Revision 2 of the ATI VGA chip has capabilities that Revision 1 does not.

To detect the presence of a VGA*WONDER* in a system, it is recommmended that the VGA BIOS ROM, which is located at address C000:0 in host memory, be interrogated for the presence of a specific code. The codes which can be checked are shown in Table 11-5. The section "Programming Examples - Identification and Configuration" contains examples on how to access BIOS data.

# Programming Examples

## Accessing Extended Registers

Extended registers on the VGA*WONDER* are accessed using two sequential I/O addresses, the first address to select index and the second address to access the data. Use 8 bit I/O instructions; 16-bit I/O is not supported during read operations.

```
MOV        DX,IO_Address         ;Load I/O address
MOV        AL,REG_INDEX          ;Load register index
OUT        DX,AL                 ;Select register
INC        DX                    ;Advance I/O address
IN         AL,DX                 ;Get the register data value
```

Extended registers of the VGA*WONDER* should be written using a single 16-bit output instruction (**OUT DX,AX**). Do not use 8-bits I/O instructions to reference the index and then the data. Code similar to the following should be used:

```
MOV        AL,Index              Load register index
MOV        AH,Data               Load register data
MOV        DX,IO_Address         Load I/O address
OUT        DX,AX                 Write to extended register
```

"IO_Address", the address of the ATI extended register bank, is stored in the BIOS ROM at address C000:0010h and is typically 1CEh. It should not be assumed that this I/O address will be the same for all ATI adapters. The following code shows how to find the proper I/O addresses for the extended register bank of the VGA*WONDER*:

```
MOV        AX,0C000H             ;Fetch segment of VGA BIOS
MOV        DS,AX                 ;Load segment register
MOV        SI,10h                ;Setup offset of register address
MOV        AX,DS:[SI]            ;Get I/O address of ATI registers
MOV        IO_Address,SI         ;Save I/O address
```

Further examples showing access to the extended register bank can be found in the example procedures Select_Page, Select_Read_Page and Select_Write_Page, and INFO.C shown below.

## Display Memory Paging

An I/O register (the Page Select register), located in the extended register bank at index B2h, is used to define which page of display memory is selected. To select a page, the desired page number is written to the appropriate bits of the Page Select register (see Figure 11-3 on page 272).

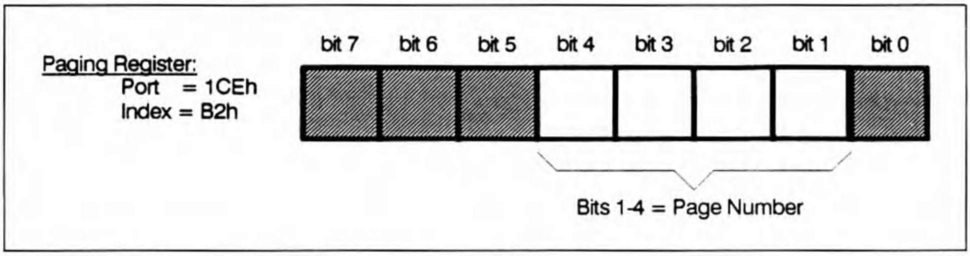


**Figure 11-3. Page Select register format**

For V4 and later versions of VGA*WONDER*, an optional enhanced paging mode is included that permits one page of display memory to be enabled for reading while a different page of display memory is enabled for writing (see Figure 11-4). Both pages reside at the same host memory address. This mode is useful when transfering data from one part of display memory to another as for on-screen to on-screen BITBLT operations (see BITBLT programming examples).

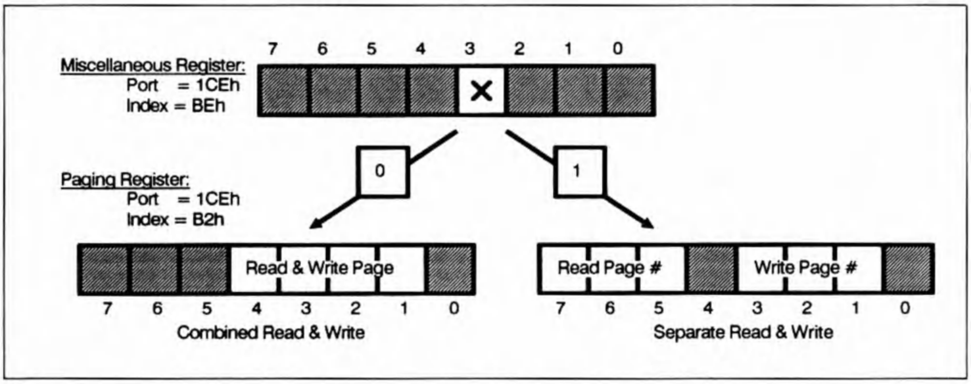


**Figure 11-4. Separate read & write paging registers**

Some ATI documentation may refer to display memory pages as memory planes which is an unfortunate choice, since it is then easily confused with VGA color planes.

For compatibility, all versions of the VGA*WONDER* default on initialization to the single page mode of operation. For those versions of the VGA*WONDER* that support dual page operation (versions V4 and V5, which use the Revision 2 VGA chip), dual page mode is enabled through the Miscellaneous register, index BEh, in the extended register bank.

The paging mechanism is illustrated in the following progamming examples. _Select_Graphics selects the display mode (mode number is obtained from include

file MODE.INC). Three procedures, _Select_Page, _Select_Read_Page and _Select_Write_Page, support display memory paging.

**Select_Mode:** This procedure is used by the program DEMO.C to initialize the board to graphics mode. It demonstrates how to obtain the extended register bank address, how to check if the board has separate read/write page capability, and how to enable separate read and write pages, if available.

**Select_Page:** This procedure demonstrates how to select a page for both read and write. It uses the flag 'Two_Pages' (initialized by the procedure Select_Mode) to determine which paging scheme to use. Note that the page select procedure will detect if the correct page is already selected to save time.

**Select Read Page and Select Write Page:** These procedures demonstrate how to select separate read and write pages by changing the corresponding nibbles in extended register B2h. Note that the other nibble is preserved when selecting either write or read page.

This module also uses the include file VGA.INC and the include file MODE.INC (included on the diskette in directory \ATI).

**Listing 11-1. File: ATI\SELECT.ASM**

```
;*********************************************************************
;* File:  SELECT.ASM                                                 *
;* Description: This module contains procedures to select mode and to *
;*       select pages.  It also initializes global variables         *
;*       according to the values in the MODE.INC include file.       *
;* Entry Points:                                                     *
;*       _Select_Graphics   - Select a graphics mode                 *
;*       _Select_Text       - Set VGA adapter into text mode         *
;*       _Select_Page       - Set page for read and write            *
;*       _Select_Read_Page  - Select read page only (ver2 chip)      *
;*       _Select_Write_Page - Select write page only (ver2 chip)     *
;* Uses:                                                             *
;*       MODE.INC      - Mode dependent constants                    *
;*       Following are modes and paths for ATI boards:               *
;*    |----- 256 colors ------|  |-- 16 colors --| 4 colors 2 colors *
;*    640x400  640x480  800x600  800x600  1024x768 1024x768 1024x768 *
;* Mode:   61h     62h      63h    6Ah(54h)  55h    67h   N/A        *
;* Path: 256COL   256COL   256COL    16COL     16COL  4COLATI   N/A  *
;*********************************************************************

     INCLUDE VGA.INC
     INCLUDE MODE.INC     ;Mode dependent constants

     PUBLIC    _Select_Graphics
     PUBLIC    _Select_Text
     PUBLIC    _Select_Page
     PUBLIC    _Select_Read_Page
     PUBLIC    _Select_Write_Page

     PUBLIC    Select_Page
     PUBLIC    Select_Read_Page
     PUBLIC    Select_Write_Page
     PUBLIC    Enable_Dual_Page
     PUBLIC    Disable_Dual_Page
```

```
     PUBLIC    Graf_Seg
     PUBLIC    Video_Height
     PUBLIC    Video_Width
     PUBLIC    Video_Pitch
     PUBLIC    Video_Pages
     PUBLIC    Ras_Buffer
     PUBLIC    Two_Pages
     PUBLIC    IO_Address
     PUBLIC    Last_Byte
;-----------------------------------------------------------------------
; Data segment variables
;-----------------------------------------------------------------------

;_DATA     SEGMENT WORD PUBLIC 'DATA'
;_DATA     ENDS

;-----------------------------------------------------------------------
; Constant definitions
;-----------------------------------------------------------------------

PAGE_SELECT     EQU  0B2h              ;Index for page select
PAGE_MASK       EQU  0E1h              ;Page mask for Read+Write select
RPAGE_MASK      EQU  01Fh              ;Read page mask
WPAGE_MASK      EQU  0F1h              ;Write page mask
DIFF_RW_PAGE    EQU  008h              ;Separate R & W pages bit in misc reg
MISC2_REG       EQU  0BEh              ;Index for misc register 2

;-----------------------------------------------------------------------
; Code segment variables
;-----------------------------------------------------------------------

_TEXT     SEGMENT BYTE PUBLIC 'CODE'

Graf_Seg        DW   0A000h            ;Graphics segment addresses
                DW   0A000h
OffScreen_Seg   DW   0A000h            ;First byte beyond visible screen
Video_Pitch     DW   SCREEN_PITCH      ;Number of bytes in one raster
Video_Height    DW   SCREEN_HEIGHT     ;Number of rasters
Video_Width     DW   SCREEN_WIDTH      ;Number of pixels in a raster
Video_Pages     DW   SCREEN_PAGES      ;Number of pages in the screen
Ras_Buffer      DB   1024 DUP (0)      ;Working buffer
R_Page          DB   0FFh              ;Most recently selected page
W_Page          DB   0FFh
RW_Page         DB   0FFh
Two_Pages       DB   0                 ;Indicate separate R & W capability

IO_Address      DW   0                 ;Address of extended registers

;***********************************************************************
;*                                                                     *
;* _Select_Graphics(HorizPtr, VertPtr, ColorsPtr)                      *
;*   Initialize VGA adapter to the graphics mode defined in MODE.INC   *
;*                                                                     *
;* Entry:                                                              *
;*   None                                                              *
;*                                                                     *
;* Returns:                                                            *
;*   VertPtr   - Vertical resolution                                   *
;*   HorizPtr  - Horizontal resolution                                 *
;*   ColorsPtr - Number of supported colors                            *
;*                                                                     *
;***********************************************************************

Arg_HorizPtr    EQU  WORD PTR [BP+4] ;Formal parameters
Arg_VertPtr     EQU  WORD PTR [BP+6] ;Formal parameters
Arg_ColorsPtr   EQU  WORD PTR [BP+8] ;Formal parameters

_Select_Graphics PROC NEAR
     PUSH BP                           ;Standard C entry point
     MOV  BP,SP
```

```
        PUSH DI                         ;Preserve segment registers
        PUSH SI
        PUSH DS
        PUSH ES

        ; Fetch address of ATI extended register

        MOV  AX,0C000h                  ;Point DS to BIOS ROM segment
        MOV  ES,AX
        MOV  SI,10h
        MOV  AX,ES:[SI]                 ;Fetch address of register
        MOV  CS:IO_Address,AX           ;Save address for later

        ; Select graphics mode

        MOV  AX,GRAPHICS_MODE           ;Select graphics mode
        INT  10h

        ; Check chip version to see if capable of separate R&W

        MOV  CS:Two_Pages,0             ;Assume version 1 (R&W not separate)
        MOV  SI,43h                     ;Offset of version byte in BIOS ROM
        CMP  BYTE PTR ES:[SI],'1'       ;Check if version 1
        JE   Pages_Set                  ;   yes, done

        ; Ensure that separate read/write page selection is enabled

        MOV  DX,CS:IO_Address           ;Fetch address of extended registers
        MOV  AL,MISC2_REG               ;Fetch index for misc register 2
        OUT  DX,AL                      ;Select misc register
        INC  DX
        IN   AL,DX                      ;Read previous value
        DEC  DX
        MOV  AH,AL                      ;Copy old value into AH
        OR   AH,DIFF_RW_PAGE            ;Set 'separate R & W page' bit
        MOV  AL,MISC2_REG               ;Fetch index for misc register 2
        OUT  DX,AX                      ;Enable separate R & W pages
        MOV  CS:Two_Pages,1             ;   no, set flag that R&W separate
Pages_Set:

        ; Reset 'last selected page'
        MOV  AL,0FFh                    ;Use 'non-existent' page number
        MOV  CS:R_Page,AL               ;Set currently selected page
        MOV  CS:W_Page,AL
        MOV  CS:RW_Page,AL

        ; Set return parameters

        MOV  SI,Arg_VertPtr                 ;Fetch pointer to vertical resolution
        MOV  WORD PTR [SI],SCREEN_HEIGHT    ;Set vertical resolution
        MOV  SI,Arg_HorizPtr                ;Fetch pointer to horizontal resolution
        MOV  WORD PTR [SI],SCREEN_WIDTH     ;Set horizontal resolution
        MOV  SI,Arg_ColorsPtr               ;Fetch pointer to number of colors
        MOV  WORD PTR [SI],SCREEN_COLORS    ;Set number of colors

        ; Clean up and return to caller

        POP  ES                         ;Restore segment registers
        POP  DS
        POP  SI
        POP  DI

        MOV  SP,BP                      ;Standard C exit point
        POP  BP
        RET
_Select_Graphics ENDP
```

```
;*********************************************************************
;                                                                    *
; Select_Page                                                        *
;    Two versions of page select are needed, one for version 1 and   *
;    another for later versions of the chip.                         *
; Entry:                                                             *
;    AL - Page number                                                *
;                                                                    *
;*********************************************************************

Select_Page     PROC NEAR
     CMP  AL,CS:RW_Page              ;Check if already selected
     JNE  SP_Go
     RET
SP_Go:
     PUSH AX
     PUSH DX
     CMP  CS:Two_Pages,0             ;Check for separate R & W
     JNZ  SP_Two_Pages
     ; Perfom page select for version 1 chip (combined R&W pages)
     MOV  AH,AL                      ;Copy page number into AH
     MOV  CS:RW_Page,AL              ;Save as most recent RW page
     MOV  CS:R_Page,0FFh             ;Invalidate R and W pages
     MOV  CS:W_Page,0FFh
     MOV  DX,CS:IO_Address           ;Fetch extended register address
     MOV  AL,PAGE_SELECT             ;Fetch page select index
     OUT  DX,AL                      ;Selet page select register
     INC  DX
     AND  AH,07h                     ;Map page number into bits 1-3
     SHL  AH,1
     IN   AL,DX                      ;Fetch current value of page select reg
     DEC  DX
     AND  AL,PAGE_MASK               ;Clear previous page setting
     OR   AH,AL                      ;Combine with new page selection
     MOV  AL,PAGE_SELECT             ;Set page select index
     OUT  DX,AX                      ;Select new page
     POP  DX
     POP  AX
     RET
     ; Perfom page select for version 2 chip (separate R&W pages)
SP_Two_Pages:
     AND  AL,07h                     ;Force page number into range
     MOV  CS:RW_Page,AL              ;Save as most recent RW page
     MOV  CS:R_Page,AL
     MOV  CS:W_Page,AL
     MOV  AH,AL                      ;Copy page number into AH
     SHL  AH,1                       ;Copy page number into bits 1-3
     ROR  AL,1                       ;Copy page number into bits 5-7
     ROR  AL,1
     ROR  AL,1
     OR   AH,AL                      ;Combine R&W pages
     MOV  DX,CS:IO_Address           ;Fetch extended register address
     MOV  AL,PAGE_SELECT             ;Fetch page select index
     OUT  DX,AX                      ;Select page number
     POP  DX
     POP  AX
     RET
Select_Page     ENDP

;*********************************************************************
;                                                                    *
; Select_Read_Page                                                   *
;    This routine will not operate properly on earlier versions of   *
;    VGA WONDER.  It will work for revision 2 and later.             *
; Entry:                                                             *
;    AL - Page number                                                *
;                                                                    *
;*********************************************************************
```

```
Select_Read_Page PROC NEAR
     CMP  AL,CS:R_Page          ;Check if already selected
     JNE  SRP_Go
     RET
SRP_Go:
     PUSH AX
     PUSH DX
     ; Select new read page
     AND  AL,07h                    ;Force page number into range
     MOV  CS:RW_Page,0FFh           ;Invalidate RW page value
     MOV  CS:R_Page,AL              ;Save new read value
     MOV  AH,AL                     ;Keep copy in AH
     ROR  AH,1                      ;Map page number into bits 5-7
     ROR  AH,1
     ROR  AH,1
     MOV  DX,CS:IO_Address          ;Fetch address of extended registers
     MOV  AL,PAGE_SELECT            ;Fetch index for page select reg
     OUT  DX,AL                     ;Select page select register
     INC  DX
     IN   AL,DX                     ;Fetch current value
     DEC  DX
     AND  AL,RPAGE_MASK             ;Clear previous page select
     OR   AH,AL                     ;Combine with new page number
     MOV  AL,PAGE_SELECT            ;Fetch index for page select reg
     OUT  DX,AX                     ;Select new page
     ; Clean up and return
     POP  DX
     POP  AX
     RET
Select_Read_Page ENDP
;*************************************************************************
;                                                                        *
; Select_Write_Page                                                      *
;    This routine will not operate properly on earlier versions of      *
;    VGA WONDER.  It will work for revision 2 and later.                 *
; Entry:                                                                 *
;    AL - Page number                                                    *
;                                                                        *
;*************************************************************************

Select_Write_Page PROC NEAR
     CMP  AL,CS:W_Page          ;Check if already selected
     JNE  SWP_Go
     RET
SWP_Go:
     PUSH AX
     PUSH DX
     ; Select new write page
     AND  AL,07h                    ;Force into range
     MOV  CS:RW_Page,0FFh           ;Invalidate RW page value
     MOV  CS:W_Page,AL              ;Save new write value
     MOV  AH,AL                     ;Keep copy in AH
     SHL  AH,1                      ;Map page number into bits 1-3
     MOV  DX,CS:IO_Address          ;Fetch address of extended registers
     MOV  AL,PAGE_SELECT            ;Fetch index for page select reg
     OUT  DX,AL                     ;Select page select register
     INC  DX
     IN   AL,DX                     ;Fetch current value
     DEC  DX
     AND  AL,WPAGE_MASK             ;Clear previous page select
     OR   AH,AL                     ;Combine with new page number
     MOV  AL,PAGE_SELECT            ;Fetch index for page select reg
     OUT  DX,AX                     ;Select new page
     ; Clean up and return
     POP  DX
     POP  AX
     RET
Select_Write_Page ENDP
```

```
;*********************************************************************
;                                                                   *
; _Select_Page(PageNumber)        Entry point from C routines       *
; _Select_Read_Page(PageNumber)                                     *
; _Select_Write_Page(PageNumber)                                    *
; Entry:                                                            *
;    PageNumber - Page number                                       *
;                                                                   *
;*********************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Page   PROC NEAR
     PUSH BP                        ;Setup frame pointer
     MOV  SP,BP
     MOV  AL,Arg_PageNumber         ;Fetch argument
     POP  BP                        ;Restore BP
     JMP  Select_Page
_Select_Page   ENDP

_Select_Read_Page   PROC NEAR
     PUSH BP                        ;Setup frame pointer
     MOV  SP,BP
     MOV  AL,Arg_PageNumber         ;Fetch argument
     POP  BP                        ;Restore BP
     JMP  Select_Read_Page
_Select_Read_Page   ENDP

_Select_Write_Page  PROC NEAR
     PUSH BP                        ;Setup frame pointer
     MOV  SP,BP
     MOV  AL,Arg_PageNumber         ;Fetch argument
     POP  BP                        ;Restore BP
     JMP  Select_Write_Page
_Select_Write_Page  ENDP

;*********************************************************************
;*                                                                  *
;* _Select_Text                                                     *
;*   Set VGA adapter to text mode                                   *
;*                                                                  *
;*********************************************************************

_Select_Text   PROC NEAR
     MOV  AX,TEXT_MODE              ;Select mode 3
     INT  10h                       ;Use BIOS to reset mode
     RET
_Select_Text   ENDP

;*********************************************************************
;*                                                                  *
;* Enable_Dual_Page                                                 *
;* Disable_Dual_Page                                                *
;*   Not supported by ATI   based boards                            *
;*                                                                  *
;*********************************************************************

Enable_Dual_Page    PROC NEAR
     RET
Enable_Dual_Page    ENDP

Disable_Dual_Page   PROC NEAR
     RET
Disable_Dual_Page   ENDP

Last_Byte:
_Text     ENDS
          END
```

## Mode 65h - 1024x768 16-Color Graphics (4-Bit Packed Pixels)

A 4-bit packed pixel memory organization does not follow any previous industry precedent and is unique to ATI boards. A separate set of example drawing routines is provided for this mode, similar to those in Chapters 7 through 9.

### *Converting (x,y) to Page:Offset*

Figure 11-1 shows the organization of display memory for this mode. Each pixel occupies four bits; each byte contains two pixels. To convert from a pixel position in x,y coordinates to a byte offset and page number in display memory, use the following equations:

Byte Offset = (Video_Pitch * y + x/2) mod 10000hex
Page Number = (Video_Pitch * y + x/2) / 10000hex
Nibble Number = x mod 2

Due to their excessive lengths, not all listings for programming examples for this memory organization are included in the text; they are available on the diskette. For each routine, the name, file, and description is included in Table 11-6.

**Table 11-6. Progamming examples for mode 65h**

| Procedure | File Name | Description |
|---|---|---|
| Write_Pixel | 16COLATI\WPIXEL.ASM | Set pixel at (x,u) to new color |
| Read_Pixel | 16COLATI\RPIXEL.ASM | Return color of pixel at (x,y) |
| Line | 16COLATI\LINE.ASM | Line drawing using Bresneham's incremental algorithm |
| Rect | 16COLATI\RECT.ASM | Draw a solid rectangle |
| Scanline | 16COLATI\SCANLINE.ASM | Fill section of scanline with solid color |
| Bitblt | 16COLATI\BITBLT.ASM | Copy block of pixels from one section of screen to another |
| Set_Cursor | 16COLATI\CURSOR.ASM | Define shape of the cursor |
| Move_Cursor | 16COLATI\CURSOR.ASM | Move cursor from one position on the screen to another |
| Remove_Cursor | 16COLATI\CURSOR.ASM | Remove cursor from the screen |
| Read_DAC | 16COLATI\DAC.ASM | Copy R,G,B values from DAC registers to a buffer |
| Load_DAC | 16COLATI\DAC.ASM | Copy R,G,B values from a buffer to DAC registers |

## Write Pixel

The logic needed to write a pixel in this mode, while simpler than for 16-color planar modes, is still more complex than for 256-color modes. Besides computing the Segment, Offset, and Page, the Mask register must also be properly set. Unlike 16-color planar modes, pixel color registers do not need to be set.

**Listing 11-2. File: 16COLATI\WPIXEL.ASM**

```
;*************************************************************************
;*                                                                       *
;* File:        WPIXEL.ASM - 4 Bit Packed Pixel Write                    *
;* Routine:     _Write_Pixel                                             *
;* Arguments:   X, Y, Color                                              *
;*                                                                       *
;*************************************************************************

        INCLUDE VGA.INC

        EXTRN   Video_Pitch:WORD
        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR

        PUBLIC  _Write_Pixel

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU     WORD PTR [BP+4]
Arg_y           EQU     WORD PTR [BP+6]
Arg_Color       EQU     BYTE PTR [BP+8]

_Write_Pixel    PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Calculate address of pixel

        MOV     AX,Arg_y                ;Convert (x,y) to Page:Offset
        MUL     CS:Video_Pitch          ;       multiply y by pitch
        MOV     CX,Arg_x                ;       fetch x
        SHR     CX,1                    ;       convert pixel to byte number
        ADD     AX,CX                   ;       add to previous product
        ADC     DX,0                    ;       and take care of carry
        MOV     DS,CS:Graf_Seg          ;Put address in DS:DI
        MOV     DI,AX
        MOV     AL,DL                   ;Copy page number into AL
        CALL    Select_Page             ;Select proper page

        ; Move value into proper nibble

        MOV     AL,Arg_Color            ;Put color in AL
        MOV     BL,0Fh                  ;Set mask assuming lower nibble
        SHR     Arg_x,1                 ;Check if odd numbered address
        JC      Value_In                ;and if so skip shifting of nibble
        SHL     AL,1                    ;Shift lower nible into upper one
        SHL     AL,1
        SHL     AL,1
        SHL     AL,1
        NOT     BL                      ;Set mask for upper nibble
Value_In:
```

```
        ; Set pixel to supplied value

        AND     AL,BL                   ;Mask to keep new bits
        NOT     BL                      ;Set mask for bits to keep
        AND     [DI],BL                 ;Preserve the other pixel in the byte
        OR      [DI],AL                 ;Combine new pixel value into byte

        ; Clean up and return

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Write_Pixel    ENDP

_TEXT   ENDS
        END
```

## Read Pixel

Read Pixel is a companion to the Write Pixel programming example. It illustrates how to convert (x,y) position to Page:Segment:Offset address, and how to access a pixel at that location. Note that in this memory organization, after a byte is obtained from display memory, each pixel must be masked and rotated into place.

Listing 11-3. File: 16COLATI\RPIXEL.ASM

```
;************************************************************************
;*                                                                      *
;* File:        RPIXEL.ASM - 4 Bit Packed Pixel Read                    *
;* Routine:     _Read_Pixel                                             *
;* Arguments:   X, Y                                                    *
;* Returns:     Color in AX                                             *
;*                                                                      *
;************************************************************************

        INCLUDE VGA.INC

        EXTRN   Video_Pitch:WORD
        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR

        PUBLIC  _Read_Pixel

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_x           EQU     WORD PTR [BP+4]
Arg_y           EQU     WORD PTR [BP+6]

_Read_Pixel     PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Calculate address of pixel and a mask
```

```
        MOV     AX,Arg_y            ;Convert (x,y) to Page:Offset
        MUL     CS:Video_Pitch      ;       multiply y by pitch
        MOV     CX,Arg_x            ;       fetch x
        SHR     CX,1                ;       convert pixel to byte number
        ADD     AX,CX               ;       add to previous product
        ADC     DX,0                ;       and take care of carry
        MOV     DS,CS:Graf_Seg      ;Put address in DS:SI
        MOV     SI,AX
        MOV     AL,DL               ;Copy page number into AL
        CALL    Select_Page         ;Select proper page

        ; Fetch the pixel value

        MOV     AL,[SI]             ;Get byte of video memory
        XOR     AH,AH               ;Clear upper byte (for return)

        ; Move pixel into lower nibble and clear the upper nibble

        SHR     Arg_x,1             ;Check if odd numbered address
        JC      Pixel_In            ;and if so skip shifting of nibble
        SHR     AL,1                ;Shift upper nible into lower one
        SHR     AL,1
        SHR     AL,1
        SHR     AL,1
Pixel_In:
        AND     AX,0Fh              ;Clear all bits except lower nibble

        ; Cleanup and return

        POP     SI                  ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP               ;Restore stack pointer
        POP     BP                  ;Restore BP
        RET
_Read_Pixel     ENDP

_TEXT   ENDS
        END
```

## Eight Simultaneous Fonts

To enable eight simultaneous fonts, extended register 5 (index B5) bit 4 must be set to 1. For each character, the attribute byte is redefined to set both color and font. The low nibble of the attribute byte defines color and the high nibble defines the font. Background color is always 0.

Font numbers do not match the set number used with BIOS service 11h. Table 11-7 has locations and corresponding set numbers (used in BIOS service 11h) for each of the valid font numbers. It should be noted that ATI boards are capable of supporting two sets of eight fonts, for a total of 16, using memory plane 3 for the second set.

**Table 11-7. Font number vs. Character set**

| Font Number | Offset in plane 2 | Character Generator Set Number |
|---|---|---|
| 0 | 0 | 0 |
| 1 | 32k | 4 |
| 2 | 8k | 1 |
| 3 | 40k | 5 |
| 4 | 16k | 2 |
| 5 | 48k | 6 |
| 6 | 24k | 3 |
| 7 | 56k | 7 |

The programming example in Listing 11-4 demonstrates how to download fonts, enable eight simultaneous fonts, and display text using all eight fonts. It starts by creating seven new fonts from the standard 8x14 and 8x8 fonts, making normal, bold, italicized, and inverse fonts from each. Each font is copied to memory plane 2 using BIOS function 11h, sub function 0 (Load Custom Character Generator). Note that the character set number is set according to the desired font number, as show in Table 11-7.

Show_Text is used to display text in each font. For each font a label is displayed, followed by 26 upper- and lower case characters, and numbers. Each font is displayed using BIOS service 13h, Write Text String, with the attribute set for the specified font.

**Listing 11-4. File: ATI\TEXT.ASM**

```
;************************************************************************
;*                                                                      *
;* File:        TEXT.ASM - Load 8 simultaneous fonts                    *
;* Description: A program to load 8 character generators and to display *
;*              eight simultaneous fonts.                               *
;*              It is assumed that color VGA monitor is attached to VGA.*
;*                                                                      *
;************************************************************************

     INCLUDE VGA.INC

_TEXT     SEGMENT BYTE PUBLIC 'CODE'
      ASSUME  CS:_TEXT, ES:NOTHING, DS:NOTHING, SS:_STACK

Text PROC FAR
     PUSH DS                         ;Save return address
     XOR  AX,AX
     PUSH AX

     MOV  AX,CS
     MOV  DS,AX                      ;Set Data seg to Code seg

     ; Force into text mode 3

     MOV  AX,3                       ;Set for mode 3, function 0
     INT  10h                        ;Use BIOS for force mode 3

     ; Fetch 8x14 character generator and load it bold as char gen 1
```

```
MOV  AX,1130h                   ;Fn=Char Gen, SubFn=Get Info
MOV  BH,2                       ;Get info on 8x14
INT  10h                        ;Use BIOS to get pointer to CG

MOV  DL,14                      ;Character height
CALL Make_Bold                  ;Convert char gen to bold

MOV  AX,1100h                   ;Fn=Char Gen, SubFn = Custom CG
MOV  CX,256                     ;Number of characters
MOV  DX,0                       ;Start with character 0
MOV  BX,0E04h                   ;Set=1, Height=12 bytes
INT  10h                        ;Let BIOS load the character generator

; Fetch 8x14 character generator and load it italicized as char gen 2

MOV  AX,1130h                   ;Fn=Char Gen, SubFn=Get Info
MOV  BH,2                       ;Get info on 8x14
INT  10h                        ;Use BIOS to get pointer to CG

MOV  DL,14                      ;Character height
CALL Make_Italics               ;Italicize the char gen

MOV  AX,1100h                   ;Fn=Char Gen, SubFn = Custom CG
MOV  CX,256                     ;Number of characters
MOV  DX,0                       ;Start with character 0
MOV  BX,0E01h                   ;Set=2, Height=12 bytes
INT  10h                        ;Let BIOS load the character generator

; Fetch 8x14 character generator and load it inverted as char gen 3

MOV  AX,1130h                   ;Fn=Char Gen, SubFn=Get Info
MOV  BH,2                       ;Get info on 8x14
INT  10h                        ;Use BIOS to get pointer to CG

MOV  DL,14                      ;Character height
CALL Make_Inverted              ;Invert the char gen

MOV  AX,1100h                   ;Fn=Char Gen, SubFn = Custom CG
MOV  CX,256                     ;Number of characters
MOV  DX,0                       ;Start with character 0
MOV  BX,0E05h                   ;Set=3, Height=12 bytes
INT  10h                        ;Let BIOS load the character generator

; Fetch 8x8 character generator and load it as char gen 4

MOV  AX,1130h                   ;Fn=Char Gen, SubFn=Get Info
MOV  BH,3                       ;Get info on 8x8
INT  10h                        ;Use BIOS to get pointer to CG

MOV  AX,1100h                   ;Fn=Char Gen, SubFn = Custom CG
MOV  CX,256                     ;Number of characters
MOV  DX,0                       ;Start with character 0
MOV  BX,0802h                   ;Set=4, Height=8 bytes
INT  10h                        ;Let BIOS load the character generator

; Fetch 8x8  character generator and load it bold as char gen 5

MOV  AX,1130h                   ;Fn=Char Gen, SubFn=Get Info
MOV  BH,3                       ;Get info on 8x8
INT  10h                        ;Use BIOS to get pointer to CG

MOV  DL,8                       ;Character height
CALL Make_Bold                  ;Convert char gen to bold

MOV  AX,1100h                   ;Fn=Char Gen, SubFn = Custom CG
MOV  CX,256                     ;Number of characters
MOV  DX,0                       ;Start with character 0
MOV  BX,0806h                   ;Set=5, Height=8 bytes
INT  10h                        ;Let BIOS load the character generator

; Fetch 8x8 character generator and load it italicized as char gen 6
```

```
     MOV  AX,1130h                  ;Fn=Char Gen, SubFn=Get Info
     MOV  BH,3                      ;Get info on 8x8
     INT  10h                       ;Use BIOS to get pointer to CG

     MOV  DL,8                      ;Character height
     CALL Make_Italics              ;Italicize the char gen

     MOV  AX,1100h                  ;Fn=Char Gen, SubFn = Custom CG
     MOV  CX,256                    ;Number of characters
     MOV  DX,0                      ;Start with character 0
     MOV  BX,0803h                  ;Set=6, Height=8 bytes
     INT  10h                       ;Let BIOS load the character generator

     ; Fetch 8x8 character generator and load it inverted as char gen 7

     MOV  AX,1130h                  ;Fn=Char Gen, SubFn=Get Info
     MOV  BH,3                      ;Get info on 8x8
     INT  10h                       ;Use BIOS to get pointer to CG

     MOV  DL,8                      ;Character height
     CALL Make_Inverted             ;Invert the char gen

     MOV  AX,1100h                  ;Fn=Char Gen, SubFn = Custom CG
     MOV  CX,256                    ;Number of characters
     MOV  DX,0                      ;Start with character 0
     MOV  BX,0807h                  ;Set=7, Height=8 bytes
     INT  10h                       ;Let BIOS load the character generator

     ; Enable multiple character fonts

     MOV  DX,1CEh                   ;Extened register bank
     MOV  AL,0B5h                   ;Index for font enable register
     OUT  DX,AL                     ;Select register
     INC  DX
     IN   AL,DX                     ;Fetch previous value
     DEC  DX
     OR   AL,10h                    ;Set enable bit
     MOV  AH,AL
     MOV  AL,0B5h
     OUT  DX,AX                     ;Enable multiple fonts

     ; Display title line

     MOV  BX,0007h                  ;Page=0, attribute=07h (font=0, color=7)
     MOV  CX,20                     ;20 characters
     MOV  DX,011Eh                  ;Row=01, column=30
     LEA  BP,Title_Msg              ;Fetch pointer to string
     MOV  AX,1300h                  ;Fn=String, SubFn=Use BL for attr.
     INT  10h                       ;Display the string

     ; Loop over fonts, displaying label and 62 character alphabet for each

     XOR  BX,BX                     ;Set counter of fonts to do
Font_Loop:
     PUSH BX                        ;Preserve counter, & put font # on stack
     SHL  BX,1                      ;Convert counter to index
     PUSH CS                        ;Put address of text on the stack
     PUSH WORD PTR CS:MSG_Ptr [BX]
     CALL Show_Text                 ;Draw next set of text
     ADD  SP,4                      ;'Pop' text address
     POP  BX                        ;Restore counter
     INC  BX                        ;Update index
     CMP  BX,8                      ;Check if all fonts done
     JL   Font_Loop                 ;Go do next font if needed

     ; Wait for a key to be pressed

     MOV  AH,00h                    ;Function return key
     INT  16h                       ;Use BIOS to get the key

     ; Disable multiple character fonts
```

```
     MOV  DX,1CEh                      ;Extened register bank
     MOV  AL,0B5h                      ;Index for font enable register
     OUT  DX,AL                        ;Select register
     INC  DX
     IN   AL,DX                        ;Fetch previous value
     DEC  DX
     AND  AL,NOT 10h                   ;Clear enable bit
     MOV  AH,AL
     MOV  AL,0B5h
     OUT  DX,AX                        ;Disable multiple fonts

     ; Clean up and exit

Show_Done:
     RET                               ;Exit
Text ENDP

;**********************************************************************
;                                                                     *
; Show Text (font, text)                                              *
;    Display 'text' as a label, followed by 62 characters of alphabet *
;    in row '2*font' with color 'font+1'                              *
;                                                                     *
;**********************************************************************

Arg_Text  EQU  DWORD PTR [BP+4]
Arg_Font  EQU  BYTE PTR [BP+8]

Alphabet  DB   'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
          DB   '0123456789'

Show_Text PROC NEAR
     PUSH BP
     MOV  BP,SP

     ; Convert font number to attribute value

     MOV  BL,Arg_Font                  ;Fetch font number
     SHL  BL,1                         ;Move into bits 4-7
     SHL  BL,1
     SHL  BL,1
     SHL  BL,1
     OR   BL,1                         ;Set initial fg color to 1
     ADD  BL,Arg_Font                  ;Use color 'font+1'

     ; Setup parameters for BIOS service call and call it to show label

     MOV  BH,0                         ;Page 0
     MOV  CX,18                        ;16 characters
     MOV  DH,Arg_Font                  ;Compute starting row
     SHL  DH,1                         ;    as 'font*2 + 3'
     ADD  DH,3
     MOV  DL,0                         ;Starting column
     LES  BP,Arg_Text                  ;Fetch pointer to string
     MOV  AX,1300h                     ;Fn=String, SubFn=Use BL for attr.
     INT  10h                          ;Display the string

    ; Alphabet

     ADD  DL,18                        ;Start in column 18
     MOV  CX,62                        ;64 characters to show
     LEA  BP,CS:Alphabet               ;Pointer to alphabet string
     MOV  AX,1300h                     ;Fn=String, SubFn=Use BL for attr.
     INT  10h                          ;Display string

     ; Clean up and exit

     POP  BP
     RET
Show_Text ENDP
```

```
;*********************************************************************
;                                                                    *
; Make_Bold                                                          *
;    Convert 8xN character genertor to bold, by shifting each byte   *
;    to the right and ORing it with the original.                    *
;                                                                    *
; Entry: DL -   Number of bytes in each character                    *
;     ES:BP - Pointer to the character generator                     *
;                                                                    *
;*********************************************************************

New_CG    DB   16*256 DUP (0)  ;Buffer for new char gen

Make_Bold PROC NEAR

     ;Setup counters

     MOV  BX,256                     ;Set counter of characters
     XOR  CH,CH                      ;Set counter of bytes
     MOV  AX,ES                      ;Set pointer to source
     MOV  DS,AX
     MOV  SI,BP
     LEA  DI,CS:New_CG               ;Set pointer to destination
     MOV  AX,CS
     MOV  ES,AX

     ; Loop over characters to change

MB_Char_Loop:
     MOV  CL,DL                      ;Set counter of bytes

     ; Loop over bytes to change

MB_Byte_Loop:
     LODSB                           ;Fetch original byte
     MOV  AH,AL                      ;Get a copy of the byte
     SHR  AL,1                       ;Shift byte to the right
     OR   AL,AH                      ;Combine bytes to make bold char
     STOSB                           ;Save new character
     LOOP MB_Byte_Loop               ;Check if all bytes done
     DEC  BX
     JG   MB_Char_Loop               ;Check if all chars done

     ; Clean up and exit

     LEA  BP,CS:New_CG               ; Set pointer to new character generator
     RET
Make_Bold ENDP

;*********************************************************************
;                                                                    *
; Make_Italics                                                       *
;    Convert 8xN character genertor to italics, by shifting each byte *
;    to the right for top, and left for bottom two bytes.            *
;                                                                    *
; Entry: DL -   Number of bytes in each character                    *
;     ES:BP - Pointer to the character generator                     *
;                                                                    *
;*********************************************************************

Make_Italics   PROC NEAR

     ;Setup counters

     MOV  BL,DL                      ;Set counter of bytes
     XOR  BH,BH
     MOV  DX,256                     ;Set counter of characters
     MOV  AX,ES                      ;Set pointer to source
     MOV  DS,AX
     MOV  SI,BP
     LEA  DI,CS:New_CG               ;Set pointer to destination
```

```
     MOV  AX,CS
     MOV  ES,AX

     ; Loop over characters to change

MI_Char_Loop:
     MOV  CX,BX                    ;Set counter of bytes
     REP  MOVSB                    ;Copy next character
     SUB  DI,BX                    ;Point at first byte
     SHR  BYTE PTR ES:[DI],1       ;Shift top two lines to the right
     SHR  BYTE PTR ES:[DI+1],1
     SHR  BYTE PTR ES:[DI+2],1
     SHR  BYTE PTR ES:[DI+3],1
     SHL  BYTE PTR ES:[DI][BX-1],1 ;Shift last two lines to the left
     SHL  BYTE PTR ES:[DI][BX-2],1
     SHL  BYTE PTR ES:[DI][BX-3],1
     SHL  BYTE PTR ES:[DI][BX-4],1
     ADD  DI,BX                    ;Point to next character
     DEC  DX                       ;Check if all chars done
     JG   MI_Char_Loop
     MOV  DL,BL                    ;Restore DL

     ; Clean up and exit

     LEA  BP,CS:New_CG             ; Set pointer to new character generator
     RET
Make_Italics   ENDP
;*************************************************************************
;                                                                        *
; Make_Inverted                                                          *
;    Convert 8xN character genertor to inverse, by inverting each        *
;    byte of the original.                                               *
;                                                                        *
; Entry: DL -   Number of bytes in each character                        *
;     ES:BP - Pointer to the character generator                         *
;                                                                        *
;*************************************************************************

Make_Inverted  PROC NEAR

     ;Setup counters

     MOV  BX,256                   ;Set counter of characters
     XOR  CH,CH                    ;Set counter of bytes
     MOV  AX,ES                    ;Set pointer to source
     MOV  DS,AX
     MOV  SI,BP
     LEA  DI,CS:New_CG             ;Set pointer to destination
     MOV  AX,CS
     MOV  ES,AX

     ; Loop over characters to change

MV_Char_Loop:
     MOV  CL,DL                    ;Set counter of bytes
     ; Loop over bytes to change

MV_Byte_Loop:
     LODSB                         ;Fetch original byte
     NOT  AL                       ;Invert the byte
     STOSB                         ;Save new character
     LOOP MV_Byte_Loop             ;Check if all bytes done
     DEC  BX
     JG   MV_Char_Loop             ;Check if all chars done

     ; Clean up and exit

     LEA  BP,CS:New_CG             ; Set pointer to new character generator
     RET
Make_Inverted  ENDP
```

```
;***********************************************************************
;                                                                      *
; Load_CG                                                              *
;    Load character generator into plane 2 at the given offset.        *
; Entry:                                                               *
;    DI      - Offset of character generator in plane 2                *
;    ES:BP   - Pointer to character generator                          *
;    DL      - Height of each character                                *
;                                                                      *
;***********************************************************************

Load_CG PROC   NEAR

     ; Enable plane 2 for write at A0000

     MOV  BX,DX                      ; Save character height
     MOV  DX,SEQUENCER_PORT          ; Address of sequencer
     MOV  AL,PLANE_ENABLE_REG        ; Plane enable reg index
     OUT  DX,AL                      ; Select register
     INC  DX
     IN   AL,DX                      ; Fetch current value
     PUSH AX                         ; Save to be restored at the end
     MOV  AL,4                       ; Select plane 2
     OUT  DX,AL

     DEC  DX
     MOV  AL,4                       ; Memory mode reg index
     OUT  DX,AL                      ; Select memory mode registers
     INC  DX
     IN   AL,DX                      ; Fetch current value
     PUSH AX                         ; Save to be restored later
     OR   AL,04h                     ; Disable odd/even
     OUT  DX,AL

     MOV  DX,GRAPHICS_CTRL_PORT      ; Address of graphics controller
     MOV  AL,MISC_REG                ; Index of misc reg
     OUT  DX,AL                      ; Select misc reg
     INC  DX
     IN   AL,DX                      ; Read current value
     PUSH AX                         ; Save to be restored later
     AND  AL,0F1h                    ; Disable odd/even and select A000
     OR   AL,04h
     OUT  DX,AL

     ; Copy character generator

     MOV  AX,ES                      ; Load DS:SI with source
     MOV  DS,AX
     MOV  SI,BP
     MOV  AX,0A000h                  ; Load ES:DI with destination
     MOV  ES,AX
     MOV  DX,BX                      ; Setup counters
     XOR  DH,DH
     MOV  BX,256
Loop1:
     MOV  CX,DX                      ; Number of bytes to copy
     REP  MOVSB                      ; Copy bytes for next character
     MOV  CX,20h                     ; Number of zero's to fill after char
     SUB  CX,DX
     REP  STOSB                      ; Fill trailing zeros
     DEC  BX                         ; Check if all characters done
     JG   Loop1                      ; Go do next char, if not all done

     ; Restore previous state

     MOV  DX,GRAPHICS_CTRL_PORT      ; Restore graphics controller
     POP  AX                         ; Get the original value
     XCHG AL,AH                      ; Setup index and data
     MOV  AL,MISC_REG
     OUT  DX,AX                      ; Restore register
```

```
      MOV  DX,SEQUENCER_PORT           ; Restore graphics controller
      POP  AX                          ; Get the original value
      XCHG AL,AH                       ; Setup index and data
      MOV  AL,4
      OUT  DX,AX                       ; Restore register

      POP  AX                          ; Get the original value
      XCHG AL,AH                       ; Setup index and data
      MOV  AL,PLANE_ENABLE_REG
      OUT  DX,AX                       ; Restore register

      RET
Load_CG ENDP

;*************************************************************************
; Data definition                                                        *
;*************************************************************************

Title_Msg DB   '8 SIMULTANEOUS FONTS'
MSG_0     DB   '0: Default Set     '
MSG_1     DB   '4: 8x14 Bold       '
MSG_2     DB   '1: 8x14 Italics    '
MSG_3     DB   '5: 8x14 Inverted   '
MSG_4     DB   '2: 8x8 Normal      '
MSG_5     DB   '6: 8x8 Bold        '
MSG_6     DB   '4: 8x8 Italics     '
MSG_7     DB   '7: 8x8 Inverted    '

MSG_Ptr   DW   OFFSET MSG_0
               DW   OFFSET MSG_1
               DW   OFFSET MSG_2
               DW   OFFSET MSG_3
               DW   OFFSET MSG_4
               DW   OFFSET MSG_5
               DW   OFFSET MSG_6
               DW   OFFSET MSG_7

_TEXT     ENDS

_STACK    SEGMENT PARA STACK 'STACK'
          DB      100h DUP(?)
_STACK    ENDS
          END
```

# Detection and Identification

When writing software that can take advantage of ATI extended features in a well-behaved manner, start by testing for the presence of the VGA*WONDER*, then check the revision level of the adapter. See Table 11-5 for the location of the ten signature bytes, and the chip version byte in the BIOS ROM. The following C program shows how to test to see if a VGA*WONDER* is present, how to identify the revision level, and how to determine other capabilities of the adapter.

**Listing 11-5. File: ATI\INFO.C**

```
/*************************************************************************/
/*                                                                       */
/* File:        INFO.C                                                   */
/* Description: This is a program to illustrate how to obtain            */
/*              information about, and state of the ATI VGA WONDER.      */
/*                                                                       */
/*************************************************************************/
```

```
#include <stdio.h>
#include <dos.h>

char ati_signature[] =              " 761295520";
char msg_chip_version[] =           "Chip Version                  : ";
char msg_BIOS_version[] =           "BIOS Version                  : ";
char msg_8bit_ROM[] =               "8 and 16 bit ROM supported    : ";
char msg_mouse_chip[] =             "Mouse chip present            : ";
char msg_interlace[] =              "Supports non-interlaced modes: ";
char msg_micro_channel[] =          "Micro Channel supported       : ";
char msg_clock_chip[] =             "Clock chip present            : ";

char msg_register1[] =              "ATI Extended register 1       : ";
char msg_EGA_emul[] =               "           EGA emulation      : ";
char msg_EGA_addr[] =               "           EGA I/O addressing : ";
char msg_register2[] =              "ATI Extended register 2       : ";
char msg_reg2_interlace[] =         "           Interlaced mode    : ";
char msg_register4[] =              "ATI Extended register 4       : ";
char msg_CGA_emul[] =               "           CGA emulation      : ";
char msg_registerB[] =              "ATI Extended register B       : ";
char msg_RAM_size[] =               "           Video RAM size     : ";
char msg_display[] =                "           Display type       : ";
char *display_type[] = {"EGA","PS/2 Mono","TTL Mono","PS/2 Color",
                         "RGB","Mulitsync","?",       "8514/A",
                         "?",  "NEC-2A",   "?",       "?",
                         "?",  "NEC-XL",   "?",       "?"};
char msg_register17[] =             "ATI Extended register 17
main()
     {
     int       i, j, k;
     union     REGS    regs;         /* Used for BIOS calls            */
     char      far     *p;           /* Used to address RAM directly */
     int       far     *q;
     int               ati_reg;
     char              value;

     /*************************************************************/
     /* Use BIOS to check if any VGA is present in the system     */
     /*************************************************************/

     /* Check for EGA                          */

     regs.h.ah = 0x12;               /* Function = Get EGA Status    */
     regs.h.bl = 0x10;
     regs.h.bh = 0x55;               /* Dummy, will change if EGA/VGA*/
     int86(0x10,&regs,&regs);
     if (regs.h.bh == 0x55)          /* Quit if BH was not changed   */
          {                     /* since this means no EGA is    */
          printf("\n...Error: EGA/VGA not present in the system\n");
          exit(0);
          }

     /* Check for VGA                                                */

     regs.h.ah = 0x1A;               /* Function: Read VGA Config.   */
     regs.h.al = 0x00;               /* Subfunction: 0               */
     int86(0x10,&regs,&regs);
     if (regs.h.al != 0x1A)          /* Quit if AL was changed since */
          {                     /* this means this is EGA BIOS  */
          printf("\n...Error: EGA in the system but VGA not found\n");
          exit(0);
          }

     /*************************************************************/
     /* Check for ATI VGA WONDER, by checking for product signature */
     /* at C000:0030.  It should be ASCII string "761295520"       */
     /*************************************************************/

     p = (char far *)0xC0000000;     /* Set pointer to VGA BIOS area */
     q = (int  far *)0xC0000000;
     for (i = 0; i < 10; i++)       /* Check signature bytes        */
```

```
     if (p[i+0x30] != ati_signature[i])
          {
          printf("\n...Error: ATI VGA WONDER not found\n");
          exit(0);
          }

/*************************************************************/
/* Display chip and BIOS version, and capabilities using data   */
/* in the BIOS code area (segment C000).                        */
/*************************************************************/

printf("%s%c\n", msg_chip_version, p[0x43]);
printf("%s%d.%d\n", msg_BIOS_version, p[0x4C], p[0x4D]);
printf("%s%s\n", msg_8bit_ROM,      (p[0x42] & 0x01) ? "Yes":"No");
printf("%s%s\n", msg_mouse_chip,    (p[0x42] & 0x02) ? "Yes":"No");
printf("%s%s\n", msg_interlace,     (p[0x42] & 0x04) ? "Yes":"No");
printf("%s%s\n", msg_micro_channel, (p[0x42] & 0x08) ? "Yes":"No");
printf("%s%s\n", msg_clock_chip,    (p[0x42] & 0x10) ? "Yes":"No");

/*************************************************************/
/* Display current status using data in the extended registers  */
/*************************************************************/

ati_reg = q[0x08];                  /* Fetch ATI ext reg adrress */

/* Display EGA emulation status                                  */

outp(ati_reg, 0xB1);                /* Select register 1         */
value = inp(ati_reg+1);             /* Fetch register 1 data     */
printf("%s%.2Xh\n", msg_register1, value);
printf("%s%s\n", msg_EGA_emul, (value & 0x02) ? "On" : "Off");
printf("%s%s\n", msg_EGA_addr, (value & 0x01) ? "On" : "Off");

/* Display interlacing status                                    */

outp(ati_reg, 0xB2);                /* Select register 2         */
value = inp(ati_reg+1);             /* Fetch register 2 data     */
printf("%s%.2Xh\n", msg_register2, value);
printf("%s%s\n", msg_reg2_interlace, (value & 0x01) ? "On" : "Off")

/* Display CGA emulation status                                  */

outp(ati_reg, 0xB4);                /* Select register 4         */
value = inp(ati_reg+1);             /* Fetch register 4 data     */
printf("%s%.2Xh\n", msg_register4, value);
printf("%s%s\n", msg_CGA_emul, (value & 0x01) ? "On"   "Off");

/* Display memory size and monitor type detected                 */

outp(ati_reg, 0xBB);                    /* Select register 2     */
value = inp(ati_reg+1);                 /* Fetch register 2 data */
printf("%s%.2Xh\n", msg_register2, value);
printf("%s%s\n", msg_RAM_size, (value & 0x20) ? ""512k'': "256k");
printf("%s%s\n", msg_display, display_type[value & 0x0F]);

/* Display status register                                      */

value = inp(0x3CF);               /* Fetch status register data */
printf("%s%.2Xh\n", msg_register17, value);
```

# 12

# Cirrus CL-GD 510, CL-GD 520 MaxLogic MaxVGA

# Introduction

Cirrus Logic started in the VGA chip business under a contract with Video Seven, designing and building a VGA chip to Video Seven specifications. Many Video Seven VGA boards carry the Cirrus logo on their VGA chips. In 1988 Video Seven started making their own chips, then merged with G-2 to form Headland Technologies (see Chapter 15).

Cirrus supplies VGA chips to PC system manufacturers. Their chips can be found on the motherboards of products such as the Intel 386SX and AT&T computers, and in many laptop computers. They can also be found on VGA add-in boards from Maxlogic, STB and Renaissance.

New features in the recently announced Cirrus CL-GD 610/CL-GD 620 chip set include additional support for lap top displays such as monochrome and color LCDs. While this chapter focuses on the 510/520 chip set, most of the information is also applicable to the 610/620 chip set. The only significant software change needed is to change the value of the Password Key used to enable access to the extended registers. For more information on this subject see the programming examples later in this chapter.

MaxVGA is equipped with 256K of display memory and supports resolutions up to 800x600 with 16 colors and 640x400 with 256 colors. It also includes emulation modes for EGA, CGA, MDA and Hercules. According to MaxLogic, tests performed by PC Labs have shown the MaxVGA to have the highest level of compatibility in VGA, CGA, MDA and Hercules modes.

MaxLogic provides drivers for popular programs such as Microsoft Windows, AutoCAD, AutoShade, GEM, Ventura, Microstation, CADKEY, Quattro, Framework, Lotus 1-2-3 and Symphony, and VESA.

# Expanded Display Modes

Table 12-1 lists the enhanced display modes of the MaxVGA. Any of these modes can be selected using the Mode Select command of the BIOS.

Table 12-1. Enhanced display modes—MaxVGA

| Mode | Type | Resolution | Colors | Memory Required | Display Type |
|---|---|---|---|---|---|
| 15h | Text | 132 col x 25 rows | mono | 256 KB | MDA |
| 16h | Text | 132 col x 44 rows | mono | 256 KB | MDA |
| 18h | Text | 132 col x 30 rows | mono | 256 KB | Multi |
| 1Eh | Text | 132 col x 25 rows | 16 | 256 KB | CGA |
| 1Fh | Text | 132 col x 25 rows | 16 | 256 KB | EGA |
| 20h | Text | 132 col x 44 rows | 16 | 256 KB | EGA |
| 22h | Text | 132 col x 30 rows | 16 | 256 KB | SuperVGA |
| 31h | Text | 100 col x 37 rows | mono | 256 KB | SuperVGA |
| 6Ah | Graphics | 800 x 600 | 16 | 256 KB | SuperVGA |
| 40h | Graphics | 720x540 | 16 | 256 KB | SuperVGA |
| 50h | Graphics | 640x400 | 256 | 256 KB | SuperVGA |
| 51h (1) | Graphics | 512x480 | 256 | 256 KB | SuperVGA |

NOTE: Some versions of the BIOS support mode 51h through a RAM resident TSR program only.

# Memory Organization

## High Resolution Text Modes

These modes utilize memory maps that are similar to those used in standard text modes (modes 0,1,2,3 and 7), except that the number of characters per line and/or the number of lines per screen is increased. Display memory is organized as shown in Figure 5-1 (see Chapter 5).

## 16-Color Graphics Modes

Memory organization for these modes resembles VGA mode 12h (640 x 480 16-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Mode 74h requires display memory paging. Display memory organization is shown in Figure 7-1; see Chapter 7 for programming examples.

## 256-Color Graphics Modes

Memory organization for modes 50h and 51h does not resemble any standard VGA mode. Each pixel occupies one byte, and bytes are spread throughout the VGA color planes as shown in the chart on page 296.

| Pixel # | Plane | Offset |
|---|---|---|
| 4 * N + 0 | 0 | N |
| 4 * N + 1 | 1 | N |
| 4 * N + 2 | 2 | N |
| 4 * N + 3 | 3 | N |

Memory organization for these modes is illustrated in Figure 12-1. For mode 50h (640x400 256 colors), each scan line occupies 160 bytes in each plane. Four consecutive pixels are addressable at each byte address (one byte per plane). Less than 64 kbytes of display memory are required per plane in these modes, so no display memory paging is required. To learn more about this memory organization, see the section titled 256 *Color Drawing* later in this chapter.

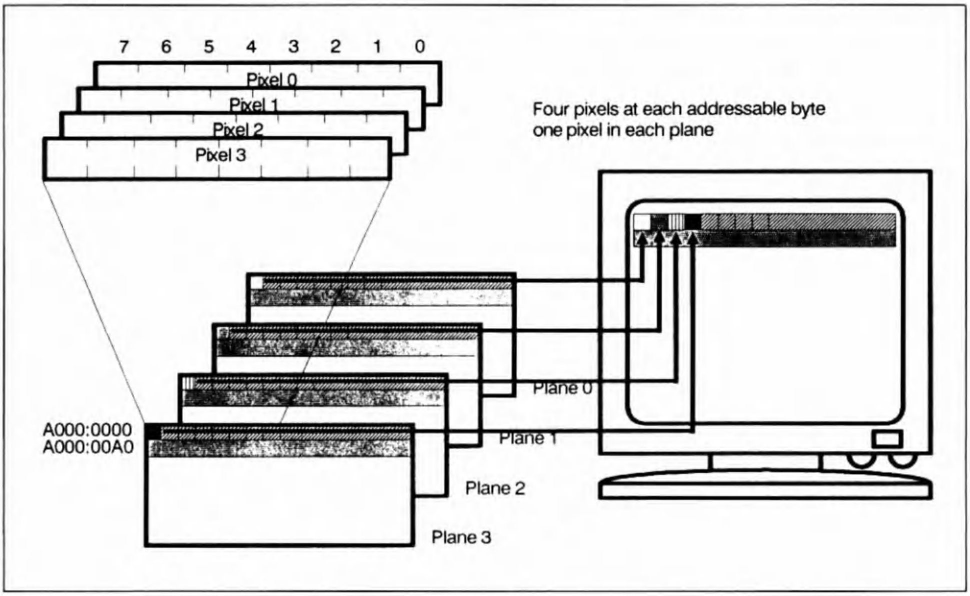


**Figure 12-1. Memory organization—mode 50h**

# Expanded Register Set

A bank of extended registers internal to the 510/520 chip set is used to access the advanced features of the adapter. The extended register bank is mapped at the same I/O address as the Sequencer, using the previously unused indexes 6 and 80h-FFh.

Most of the registers in the extended bank have read and write capability. Table 12-2 shows the extended register set of the 510/520 chip set.

When accessing the extended register bank, it is recommended that the following rules be observed:

- Before the first access to extended registers, enable them by writing the proper password value (ECh) to index 6 of the Sequencer (address 3C4h).
- Disable access to the extended registers whenever possible by writing the proper password value (CEh) to index 6 of the Sequencer (address 3C4h).
- Always restore the extended registers when done, or at least set them to "non-disruptive" values (generally zero). BIOS mode select does not always reset the extended registers.

Different password values are used for the newer 610/620 chip set. The proper unlocking password value can be obtained for either chip set from the Identification Register (CRTC index 1Fh). CRTC index 0Ch must first be cleared to zero before the Identification Register is accessed. To learn more about the Identification register and Extension Control register see section titled *Detection and Identification* at the end of this chapter.

**Table 12-2. Extended registers—Cirrus 510, 520**

| I/O Address | Index | Register |
|---|---|---|
| 3D4h/3B4h | 1Fh | Identification (value ECh XORed with index 0Ch in CRTC) (read only) |
| 3C4h | 06h | Extension Control |
| | 80h | Miscellaneous Control 1 |
| | 81h | Graphics 1 position |
| | 82h | Graphics 2 position |
| | 83h | Attibute Controller Index |
| | 84h | Write Control |
| | 85h | Timing Control |
| | 86h | Bandwidth Control |
| | 87h | Miscellaneous Control 2 |
| | 88h | Horizontal synch skew |
| | 89h | CGA, HGC Font Control |
| | 8Ah | Reserved |
| | 8Bh | Screen B preset row scan |
| | 8Ch | Screen B start address high |
| | 8Dh | Screen B start address low |
| | 8Eh | Version code (read only) |
| | 8Fh | Version code (read only) |
| | 90h | Vertical retrace start |
| | 91h | Vertical retrace end |

**Table 12-2. Extended registers—Cirrus 510, 520** *(continued)*

| I/O Address | Index | Register |
|---|---|---|
| | 92h | Lightpen high |
| | 93h | Lightpen low |
| | 94h | Pointer pattern address high |
| | 95h | Cursor height adjust |
| | 96h | Caret width |
| | 97h | Caret height |
| | 98h | Caret horizontal position high |
| | 99h | Caret horizontal position low |
| | 9Ah | Caret vertical position high |
| | 9Bh | Caret vertical position low |
| | 9Ch | Pointer horizontal position high |
| | 9Dh | Pointer horizontal position low |
| | 9Eh | Pointer vertical position high |
| | 9Fh | Pointer vertical position low |
| | A0h | Graphics controler memory latch 0 |
| | A1h | Graphics controler memory latch 1 |
| | A2h | Graphics controler memory latch 2 |
| | A3h | Graphics controler memory latch 3 |
| | A4h | Clock select |
| | A5h | Cursor (caret and pointer) attribute |
| | A6h | Internal switch source |
| | A7h | Status switch control |
| | A8h | NMI mask 1 |
| | A9h | NMI mask 2 |
| | AAh | Reserved |
| | ABh | NMI status 1 (read only) |
| | ACh | NMI status 2 (read only) |
| | ADh | 256-color mode page control |
| | AEh | NMI data cache (four 24-bit words) (read only) |
| | AFh | Active adapter state |
| | B0h - BFh | Scratch registers |
| | C0h - FFh | Reserved |

Note: Extension registers 80h to FFh must be enabled for write using Extension Control.

# Programming Examples

## 256-Color Drawing

Drawing algorithms for these modes are unlike any of the drawing algorithms used for other video boards. A separate directory of the diskette, named 256COLCI, contains drawing routines for this mode. Due to the overall length of these examples, only examples Write_Pixel and Read_Pixel are shown in the text (Listings 12-1 and 12-2).

**Listing 12-1. File: 256COLCI\WPIXEL.ASM**

```
;************************************************************************
;*                                                                      *
;* File:        WPIXEL.ASM - 8 Bit Pixel Write - Alternating planes     *
;* Routine:     _Write_Pixel                                            *
;* Arguments:   X, Y, Color                                             *
;*                                                                      *
;************************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pitch:WORD

        PUBLIC  _Write_Pixel
        PUBLIC  Select_Plane

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_X           EQU     WORD PTR [BP+4]
Arg_Y           EQU     WORD PTR [BP+6]
Arg_Color       EQU     BYTE PTR [BP+8]

_Write_Pixel    PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Convert x,y pixel address to Offset and Plane enable

        MOV     AX,Arg_Y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;   multiply by width in bytes
        MOV     CX,Arg_X                ;   add x/4 to compute offset
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        MOV     DS,CS:Graf_Seg          ;Put new address in DS:SI
        MOV     DI,AX

        MOV     AX,Arg_X                ;Fetch x coordinate
        CALL    Select_Plane            ;Enable plane according to X

        ; Set pixel to supplied value

        MOV     AL,Arg_Color            ;Fetch color to use
        MOV     [DI],AL                 ;Set the pixel

        ; Clean up and return
```

```
        POP     SI                          ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                       ;Restore stack pointer
        POP     BP                          ;Restore BP
        RET
_Write_Pixel    ENDP

;*********************************************************************
; Plane_Enable                                                       *
;       Enable plane according to x coorinate (plane = x mod 4)      *
;                                                                    *
; Entry: AL - x coordinate                                           *
;*********************************************************************

Select_Plane    PROC NEAR
        PUSH    AX
        PUSH    CX
        PUSH    DX
        ;Convert plane number to mask (for write enable)
        AND     AL,3h                       ;Compute x mod 4 to get plane number
        MOV     CH,AL                       ;Save plane number for later
        MOV     CL,AL                       ;Use plane number to get rotate count
        MOV     AH,01h                      ;Convert plane number to mask
        SHL     AH,CL
        ;Enable plane for write
        MOV     DX,SEQUENCER_PORT           ;Fetch address of sequencer
        MOV     AL,PLANE_ENABLE_REG         ;Index of plane enable register
        OUT     DX,AX                       ;Enable plane for write
        ;Select plane for read
        MOV     DX,GRAPHICS_CTRL_PORT       ;Fetch address of graphics controller
        MOV     AL,READ_PLANE_REG           ;Index of read plane enable register
        MOV     AH,CH                       ;Fetch plane number
        OUT     DX,AX                       ;Select plane for read
        ;Cleanupa and return
        POP     DX
        POP     CX
        POP     AX
        RET
Select_Plane    ENDP

_TEXT   ENDS
        END
```

## Listing 12-2. File: 256COLCI\RPIXEL.ASM

```
;*********************************************************************
;*                                                                   *
;* File:        RPIXEL.ASM - 8 Bit Packed Pixel Read                 *
;* Routine:     _Read_Pixel                                          *
;* Arguments:   X, Y                                                 *
;* Returns:     Color in AX                                          *
;*                                                                   *
;*********************************************************************

        INCLUDE VGA.INC

        EXTRN   Graf_Seg:WORD
        EXTRN   Select_Page:NEAR
        EXTRN   Video_Pitch:WORD
        EXTRN   Select_Plane:NEAR

        PUBLIC  _Read_Pixel

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Arg_X           EQU     WORD PTR [BP+4]
Arg_Y           EQU     WORD PTR [BP+6]
```

```
_Read_Pixel     PROC NEAR
        PUSH    BP                      ;Preserve BP
        MOV     BP,SP                   ;Preserve stack pointer

        PUSH    ES                      ;Preserve segment and index registers
        PUSH    DS
        PUSH    DI
        PUSH    SI

        ; Convert x,y pixel address to Offset and Plane number

        MOV     AX,Arg_Y                ;Fetch y coordinate
        MUL     CS:Video_Pitch          ;   multiply by width in bytes
        MOV     CX,Arg_X                ;   add x/4 to compute offset
        SHR     CX,1
        SHR     CX,1
        ADD     AX,CX
        MOV     DS,CS:Graf_Seg          ;Put new address in DS:SI
        MOV     SI,AX

        MOV     AX,Arg_X                ;Fetch x coordinate
        CALL    Select_Plane            ;Enable plane for read

        ; Fetch the pixel value

        MOV     AL,[SI]                 ;Get byte of video memory
        XOR     AH,AH                   ;Clear upper byte (for return)

        ; Cleanup and return

        POP     SI                      ;Restore segment and index registers
        POP     DI
        POP     DS
        POP     ES

        MOV     SP,BP                   ;Restore stack pointer
        POP     BP                      ;Restore BP
        RET
_Read_Pixel     ENDP

_TEXT   ENDS
        END
```

## Graphics Cursor Control

The Cirrus 510,520 VGA chip set includes hardware support for a graphics cursor that can significantly reduce the processor overhead required for cursor control. Its usefulness is limited, however, since the hardware cursor cannot be used in 256-color modes. Figure 12-2 illustrates the operation of the hardware graphics cursor. Seven registers in the extended register bank are involved in the definition and control of the graphics cursor.

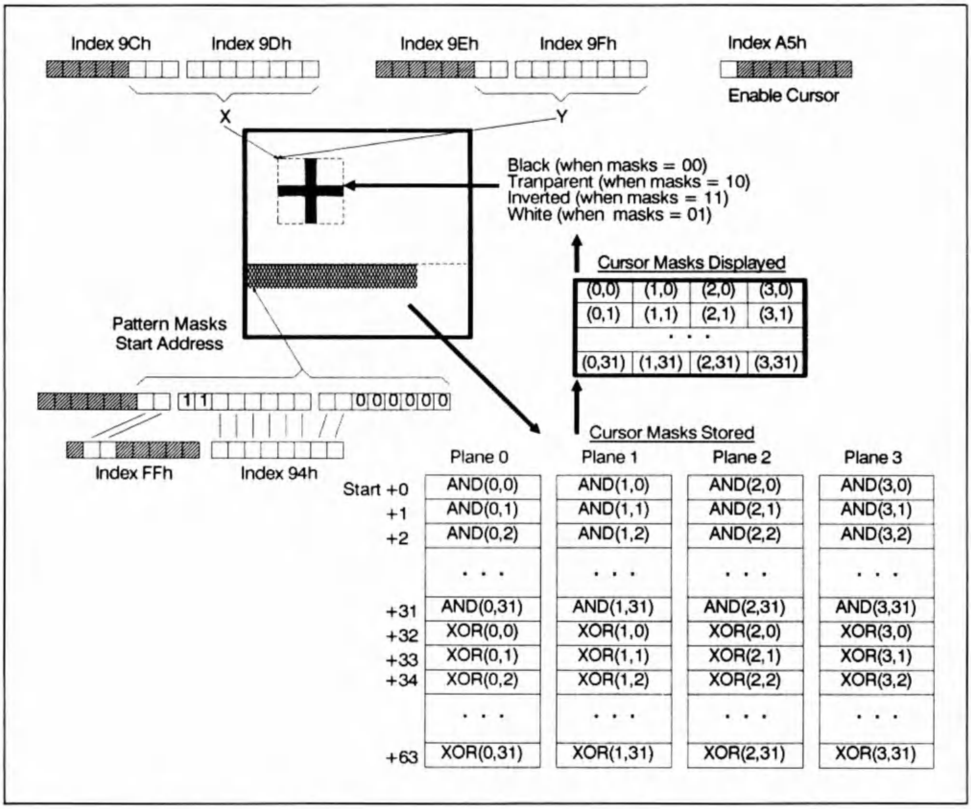


**Figure 12-2. Hardware cursor registers**

Hardware cursors operate differently than software cursors. Since the cursor is drawn as an overlay on the screen, there is never any need to save background data in the cursor area. The cursor is defined by two monochrome bitmaps, or masks, which correspond to the conventional AND and XOR masks used for software cursors (for more on software cursors see our previous text, *Programmer's Guide to the EGA/VGA*).

Cursor pattern data must be loaded into offscreen display memory in a scrambled format. Figure 12-2 shows cursor pattern locations. Each row of cursor, for each mask, is defined by four bytes of pattern (32 bits for each 32-pixel row of the cursor), each byte in a separate plane. Each byte defines 8 pixels, with the most significant bit corresponding to left-most pixel. Bytes for AND mask are in the first 32 bytes (of each plane), and for XOR mask in next 32 bytes (of each plane). Each byte in Figure 12-2 is labeled as (column, row) to indicate which byte in the cursor it controls.

The programming example in Listing 12-3 illustrates how to define cursor shape and how to move the cursor around the screen. Three procedures are provided.

**Set_Cursor** is used to store AND and XOR masks into off-screen display memory, and how to enable the cursor display.

**Move_Cursor** is used to determine where the cursor is displayed.

**Remove_Cursor** disables the cursor display.

Listing 12-3. File: CIRRUS\HWCURSOR.ASM

```
;*********************************************************************
;*                                                                   *
;* File:         HWCURSOR.ASM                                        *
;* Description: This module contains procedures to demonstrate use of a *
;*              hardware cursor.  It defines cursor shape, moves     *
;*              cursor around the screen, and removes cursor.        *
;*                                                                   *
;* Entry Points:                                                     *
;*                                                                   *
;*              _Set_Cursor                                          *
;*              _Move_Cursor                                         *
;*              _Remove_Cursor                                       *
;*                                                                   *
;* Uses:                                                             *
;*              _Select_Page                                         *
;*              _Graf_Seg                                            *
;*              _Video_Height                                        *
;*              _Video_Pitch                                         *
;*                                                                   *
;*********************************************************************

        INCLUDE VGA.INC
        INCLUDE MODE.INC                ;Mode dependent constants

        EXTRN   Graf_Seg:WORD
        EXTRN   Video_Pitch:WORD
        EXTRN   Video_Height:WORD
        EXTRN   Video_Colors:WORD
        EXTRN   _BitBlt:NEAR
        EXTRN   Select_Page:NEAR

        PUBLIC  _Set_Cursor
        PUBLIC  _Move_Cursor
        PUBLIC  _Remove_Cursor

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

;---------------------------------------------------------------------
; Common cursor definitions
;---------------------------------------------------------------------

;*********************************************************************
;*                                                                   *
;* _Set_Cursor(AND_Mask, XOR_Mask, FG_Color, BG_Color)               *
;*      This procedure saves the two masks in the offscreen memory   *
;*      according to Video 7 schema. Colors are ignored.             *
;*                                                                   *
;* Entry:                                                            *
;*      AND_Mask - 4x32 bytes with AND mask                          *
;*      XOR_Mask - 4x32 bytes with XOR mask                          *
;*      BG_Color - Foreground color                                  *
;*      FG_Color - Background color                                  *
;*                                                                   *
;*********************************************************************

Arg_AND_Mask    EQU     WORD PTR [BP+4] ;Formal parameters
Arg_XOR_Mask    EQU     WORD PTR [BP+6]
```

```
Arg_BG_Color    EQU     BYTE PTR [BP+8]
Arg_FG_Color    EQU     BYTE PTR [BP+10]

_Set_Cursor     PROC NEAR
        ; Jump to software cursor routines if 256-color mode

        CMP     WORD PTR CS:Video_Colors,256
        JNE     Set_HW_Cursor
        JMP     Set_SW_Cursor
Set_HW_Cursor:

        ; Save registers

        PUSH    BP                      ;Standard high-level entry
        MOV     BP,SP

        PUSH    SI                      ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Enable all planes and all bits for write

        MOV     DX,SEQUENCER_PORT       ;Address of Sequencer
        MOV     AX,PLANE_ENABLE_REG+0F00h
        OUT     DX,AX                   ;Enable all planes for write
        MOV     DX,GRAPHICS_CTRL_PORT   ;Address of Graphics controller
        MOV     AX,SR_ENABLE_REG+0000h  ;Index and value for Set/Reset
        OUT     DX,AX                   ;Disable set/reset
        MOV     AX,BIT_MASK_REG+0FF00h  ;Index and value for bit mask reg
        OUT     DX,AX                   ;Enable all 8 bits for write

        ; Set offset for cursor mask location

        MOV     DX,SEQUENCER_PORT       ;Address of extended register bank
        MOV     AL,94h                  ;Index of pointer pattern address reg
        MOV     AH,0FFh                 ;Indicate last pointer
        OUT     DX,AX                   ;Set the new address

        ; Copy masks to off-screen memory

        MOV     ES,CS:Graf_Seg          ;Segment of display memory
        MOV     DI,-64                  ;Offset is 64 bytes before end of page
        MOV     SI,Arg_AND_Mask         ;Address of AND mask

        MOV     CX,32                   ;Initialize counter
Copy_AND_Loop:
        MOV     AX,0102h                ;Enable first plane for write
        OUT     DX,AX
        MOV     BH,DS:[SI+0]            ;Fetch next value of AND mask
        MOV     ES:[DI],BH              ;Place it in display memory
        SHL     AH,1                    ;Enable next plane
        OUT     DX,AX
        MOV     BH,DS:[SI+1]            ;Fetch next value of AND mask
        MOV     ES:[DI],BH              ;Place it in display memory
        SHL     AH,1                    ;Enable next plane
        OUT     DX,AX
        MOV     BH,DS:[SI+2]            ;Fetch next value of AND mask
        MOV     ES:[DI],BH              ;Place it in display memory
        SHL     AH,1                    ;Enable next plane
        OUT     DX,AX
        MOV     BH,DS:[SI+3]            ;Fetch next value of AND mask
        MOV     ES:[DI],BH              ;Place it in display memory
        INC     DI
        ADD     SI,4
        LOOP    Copy_AND_Loop

        MOV     CX,32                   ;Initialize counter
        MOV     SI,Arg_XOR_Mask         ;Fetch pointer to XOR mask
Copy_XOR_Loop:
        MOV     AX,0102h                ;Enable first plane for write
```

```
        OUT     DX,AX
        MOV     BH,DS:[SI+0]            ;Fetch next value of XOR mask
        MOV     ES:[DI],BH              ;Place it in display memory
        SHL     AH,1                    ;Enable next plane
        OUT     DX,AX
        MOV     BH,DS:[SI+1]            ;Fetch next value of XOR mask
        MOV     ES:[DI],BH              ;Place it in display memory
        SHL     AH,1                    ;Enable next plane
        OUT     DX,AX
        MOV     BH,DS:[SI+2]            ;Fetch next value of XOR mask
        MOV     ES:[DI],BH              ;Place it in display memory
        SHL     AH,1                    ;Enable next plane
        OUT     DX,AX
        MOV     BH,DS:[SI+3]            ;Fetch next value of XOR mask
        MOV     ES:[DI],BH              ;Place it in display memory
        INC     DI
        ADD     SI,4
        LOOP    Copy_XOR_Loop

        ; Set cursor postion at x=0 and y=last_line+1

        MOV     DX,3C4h                 ;Address of extended registers
        MOV     AL,9Ch                  ;Index of cursor x
        XOR     AH,AH                   ;Value
        OUT     DX,AX                   ;Set hi-x to 0
        INC     AL
        OUT     DX,AX                   ;Set lo-x to 0
        INC     AL
        MOV     BX,CS:Video_Height      ;Fetch number of last_line+1
        MOV     AH,BH
        OUT     DX,AX                   ;Set hi-y
        INC     AL
        MOV     AH,BL
        OUT     DX,AX                   ;Set lo-y

        ; Enable the cursor (will be below last on-screen line)

        MOV     DX,3C4h                 ;Address of extended registers
        MOV     AL,0A5h                 ;Index of cursor attr reg
        OUT     DX,AL                   ;Select cursor attr reg
        INC     DX
        IN      AL,DX                   ;Fetch current value
        OR      AL,80h                  ;Turn cursor on
        OUT     DX,AL

        ; Clean up and return

        POP     DS                      ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI

        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
_Set_Cursor     ENDP

;*********************************************************************
;*                                                                   *
;* _Move_Cursor(Curs_X, Curs_Y)                                      *
;*      This procedure is used to move the cursor from one           *
;*      location to another, by setting new cursor position registers. *
;*                                                                   *
;*********************************************************************

Arg_Curs_X      EQU     WORD PTR [BP+4] ;Formal parameters
Arg_Curs_Y      EQU     WORD PTR [BP+6]

_Move_Cursor    PROC    NEAR
        ; Jump to software cursor routines if 256-color mode
```

```
        CMP     WORD PTR CS:Video_Colors,256
        JNE      Move_HW_Cursor
        JMP      Move_SW_Cursor
Move_HW_Cursor:

        ; Save registers

        PUSH    BP                        ;Standard high-level entry
        MOV     BP,SP
        SUB     SP,4

        PUSH    SI                        ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Set cursor position

        MOV     DX,3C4h                   ;Address of extended registers
        MOV     AL,9Ch                    ;Index of first cursor pos reg
        MOV     BX,Arg_Curs_x             ;Fetch cursor x

        MOV     AH,BH                     ;Set hi-x
        OUT     DX,AX
        INC     AL

        MOV     AH,BL                     ;Set lo-x
        OUT     DX,AX
        INC     AL

        MOV     BX,Arg_Curs_y             ;Fetch cursor y
        MOV     AH,BH                     ;Set hi-y
        OUT     DX,AX
        INC     AL

        MOV     AH,BL                     ;Set lo-y
        OUT     DX,AX

        ; Clean up and return

        POP     DS                        ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI

        MOV     SP,BP                     ;Restore stack
        POP     BP
        RET
_Move_Cursor    ENDP

;************************************************************************
;*                                                                      *
;* _Remove_Cursor                                                       *
;*      This procedure is used to remove the cursor from the screen     *
;*      by disabling cursor display.                                    *
;*                                                                      *
;************************************************************************

_Remove_Cursor  PROC NEAR
        : Jump to software cursor routines if 256-color mode

        CMP     WORD PTR CS:Video_Colors,256
        JNE     Remove_HW_Cursor
        JMP     Remove_SW_Cursor
Remove_HW_Cursor:

        ; Save registers

        PUSH    BP                        ;Standard high-level entry
        MOV     BP,SP
```

```
        PUSH    SI                          ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Disable the cursor

        MOV     DX,3C4h                     ;Address of extended registers
        MOV     AL,0A5h                     ;Index of cursor attr reg
        OUT     DX,AL                       ;Select cursor attr reg
        INC     DX
        IN      AL,DX                       ;Fetch current value
        AND     AL,NOT 80h                  ;Turn cursor off
        OUT     DX,AL

        ; Clean up and return

        POP     DS                          ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI

        MOV     SP,BP                       ;Restore stack
        POP     BP
        RET
_Remove_Cursor  ENDP

;--------------------------------------------------------------------------
;--------------- Software Cursor Routines ---------------------------------
;--------------------------------------------------------------------------

;--------------------------------------------------------------------------
; Common cursor definitions
;--------------------------------------------------------------------------

CUR_WIDTH       EQU     32
CUR_HEIGHT      EQU     32

AND_OFFSET      EQU     0                   ;Save area offsets in off-screen area
XOR_OFFSET      EQU     CUR_WIDTH
CUR_OFFSET      EQU     2*CUR_WIDTH
MIX_OFFSET      EQU     4*CUR_WIDTH

Last_Cursor_x   DW      0                   ;Code segment variables
Last_Cursor_y   DW      0
Save_Area_y     DW      0
Save_Offset     DW      0

;**************************************************************************
;*                                                                        *
;* _Set_Cursor(AND_Mask, XOR_Mask, FG_Color, BG_Color)                    *
;*      This procedure will expand the two cursor masks into              *
;*      color.   Normally the masks should be stored after the            *
;*      last visible scan line (global parameter 'Video_Height',          *
;*      however in this demo, the cursor masks and the 'save buffer'      *
;*      will be stored immediately above the last line.  This is done     *
;*      so that the reader can clearly see the AND mask, the XOR mask,    *
;*      and the area under the cursor in 'save buffer'.                   *
;*                                                                        *
;* Entry:                                                                 *
;*      AND_Mask - 4x32 bytes with AND mask                               *
;*      XOR_Mask - 4x32 bytes with XOR mask                               *
;*      BG_Color - Foreground color                                       *
;*      FG_Color - Background color                                       *
;*                                                                        *
;**************************************************************************

Arg_AND_Mask    EQU     WORD PTR [BP+4] ;Formal parameters
Arg_XOR_Mask    EQU     WORD PTR [BP+6]
Arg_BG_Color    EQU     BYTE PTR [BP+8]
Arg_FG_Color    EQU     BYTE PTR [BP+10]
```

```
Set_SW_Cursor   PROC NEAR
        PUSH    BP                      ;Standard high-level entry
        MOV     BP,SP
        SUB     SP,2

        PUSH    SI                      ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Fill with background

        MOV     CX,0                    ;Set x to start of save area
        MOV     AX,CS:Video_Height      ;Set y to below last line on the screen
        ;!!!!!!!!!!!! The next line should be removed    !!!!!!!!!!!!!!!!!!!
        ;!!!!!!!!!!!! if you do not want to see the save !!!!!!!!!!!!!!!!!!!
        ;!!!!!!!!!!!! regions on the screen              !!!!!!!!!!!!!!!!!!!
        MOV     AX,0                    ;Make visible for demo !!!!!!!!!!!!!
        MOV     CS:Save_Area_y,AX       ;Save y for other cursor procs
        MUL     CS:Video_Pitch          ;   multiply y by width in bytes
        ADD     AX,CX                   ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits

        MOV     DI,AX                   ;Set DI to save area offset
        MOV     CS:Save_Offset,AX       ;Save offset for later
        MOV     ES,CS:Graf_Seg          ;Set segment to graphics segment
        MOV     AL,DL                   ;Copy page number into AL
        CALL    Select_Page             ;Select page for save area

        MOV     DX,CUR_HEIGHT           ;Number of scanlines to do
        MOV     BX,CS:Video_Pitch       ;Calculate scan-to-scan increment
        SUB     BX,CUR_WIDTH*2

        MOV     AL,Arg_BG_Color         ;Fetch background color
        MOV     AH,AL                   ;Copy color into AH
Fill_Background:
        MOV     CX,CUR_WIDTH            ;Number of words of AND & XOR mask
        REP     STOSW                   ;Fill next row of AND and XOR masks
        ADD     DI,BX                   ;Point to next scanline (assumes in
                                        ;one page!!!).

        DEC     DX                      ;Check if all scanlines done
        JG      Fill_Background         ;Go do next scanline if not done

        ; Change foreground bits for the AND mask save area

        MOV     DL,CUR_HEIGHT           ;Initialize raster counter
        MOV     DH,Arg_FG_Color         ;Fetch foreground color
        MOV     DI,CS:Save_Offset       ;Get pointer to save area
        MOV     SI,Arg_AND_Mask         ;Fetch pointer to AND-mask section
        ADD     BX,CUR_WIDTH            ;Adjust scan-to-scan increment

Set_AND_FG:
        LODSW                           ;Fetch next 16 bits from the mask
        XCHG    AL,AH                   ;Swap byte to compensate for 80xx mem
        MOV     CX,16                   ;Number of bits to do
AND_Bit_Loop:
        SHL     AX,1                    ;Move next bit into carry
        JNC     AND_Done                ;Do not change if bit not set
        MOV     ES:[DI],DH              ;Set pixel to fg color if bit set
AND_Done:
        INC     DI                      ;Update pointer
        LOOP    AND_Bit_Loop            ;If not all 16 bits done do next bit
        XOR     BX,8000h                ;Toggle high bit of BX to check if
        JS      Set_AND_FG              ; both words have been done

        ADD     DI,BX                   ;Point to next scanline
        DEC     DL                      ;Check if all scanlines done
        JG      Set_AND_FG              ;Go do next scanline if not done

        ; Change foreground bits for the XOR mask save area
```

```
        MOV     DL,CUR_HEIGHT           ;Initialize raster counter
        MOV     DH,Arg_FG_Color         ;Fetch foreground color
        MOV     DI,CS:Save_Offset       ;Get pointer to save area
        ADD     DI,XOR_OFFSET           ;Advance pointer to XOR-mask section
        MOV     SI,Arg_XOR_Mask         ;Fetch pointer to XOR-mask

Set_XOR_FG:
        LODSW                           ;Fetch next 16 bits from the mask
        XCHG    AL,AH                   ;Swap byte to compensate for 80xx mem
        MOV     CX,16                   ;Number of bits to do
XOR_Bit_Loop:
        SHL     AX,1                    ;Move next bit into carry
        JNC     XOR_Done                ;Do not change if bit not set
        MOV     ES:[DI],DH              ;Set pixel to fg color if bit set
XOR_Done:
        INC     DI                      ;Update pointer
        LOOP    XOR_Bit_Loop            ;If not all 16 bits done do next bit
        XOR     BX,8000h                ;Toggle high bit of BX to check if
        JS      Set_XOR_FG              ; both words have been done

        ADD     DI,BX                   ;Point to next scanline
        DEC     DL                      ;Check if all scanlines done
        JG      Set_XOR_FG              ;Go do next scanline if not done

        ; Set 'last cursor' to save area (this is needed for first
        ; call to Move_Cursor procedure, since first thing done in there
        ; is restore area under 'last cursor' position)

        MOV     AX,CS:Save_Area_y       ;Fetch save area y
        MOV     CS:Last_Cursor_y,AX     ;Set last cursor y
        MOV     CS:Last_Cursor_x,CUR_OFFSET  ;Set last cursor x

        ; Clean up and return

        POP     DS                      ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI

        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
Set_SW_Cursor   ENDP

;************************************************************************
;*                                                                      *
;* _Move_Cursor(Curs_X, Curs_Y)                                         *
;*      This procedure is used to move the cursor from one              *
;*      location to another. The cursor move is performed using the     *
;*      following steps:                                                *
;*              1 - Check if new cursor is outside 'cursor block'       *
;*              2 - If outside 'cursor block' restore area under        *
;*                  previous block.                                     *
;*                  Save area under new block.                          *
;*              3 - Copy saved are into cursor build area (both save and*
;*                  build areas are normally off-screen).               *
;*              4 - Combine AND and XOR masks with build area.          *
;*              5 - Copy build area to where new cursor should be (this *
;*                  in most cases overwrites the old cursor).           *
;*      The 'build area' is a rectangle twice the size of the cursor.   *
;*      It is used to eliminate flicker for small movement of the       *
;*      cursor, since cursor may not need to be erased if it moves      *
;*      only by a few pixels.                                           *
;*                                                                      *
;* Entry:                                                               *
;*      Curs_X - Position of the new cursor                             *
;*      Curs_Y                                                          *
;*                                                                      *
;************************************************************************

Arg_Curs_X      EQU     WORD PTR [BP+4] ;Formal parameters
```

```
Arg_Curs_Y      EQU     WORD PTR [BP+6]

Curs_X          EQU     WORD PTR [BP-2]
Curs_Y          EQU     WORD PTR [BP-4]

Move_SW_Cursor PROC NEAR
        PUSH    BP                      ;Standard high-level entry
        MOV     BP,SP
        SUB     SP,4

        PUSH    SI                      ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Check if new area needs to be saved

        MOV     AX,Arg_Curs_x           ;Fetch new x
        AND     AX,NOT(CUR_WIDTH-1)     ;Round to nearest buffer block
        MOV     BX,Arg_Curs_y           ;Fetch new y
        AND     BX,NOT(CUR_HEIGHT-1)    ;Round to nearest buffer block

        CMP     AX,CS:Last_Cursor_x     ;Check if x moved into next block
        JNE     Cursor_New_Block
        CMP     BX,CS:Last_Cursor_y     ;Check if y moved into next block
        JNE     Cursor_New_Block
        JMP     Build_Cursor

        ; For new block call to remove old cursor, then use_BitBlt
        ; to save block under next cursor location into the save area

Cursor_New_Block:
        CALL    _Remove_Cursor          ;Restore last location
        MOV     AX,Arg_Curs_x           ;Fetch new x
        AND     AX,NOT(CUR_WIDTH-1)     ;Round to nearest buffer block
        MOV     CS:Last_Cursor_x,AX     ;Save as 'last x'
        MOV     AX,Arg_Curs_y           ;Fetch new y
        AND     AX,NOT(CUR_HEIGHT-1)    ;Round to nearest buffer block
        MOV     CS:Last_Cursor_y,AX     ;Save as 'last y'

        MOV     AX,2*CUR_HEIGHT         ;Push width and height
        PUSH    AX
        MOV     AX,2*CUR_WIDTH
        PUSH    AX
        PUSH    CS:Save_Area_y          ;Push x and y of destination
        MOV     AX,CUR_OFFSET
        PUSH    AX
        PUSH    CS:Last_Cursor_y        ;Push x and y of source
        PUSH    CS:Last_Cursor_x
        CALL    _BitBlt
        ADD     SP,12

        ; Use _BitBlt to copy save area into build area

Build_Cursor:
        MOV     AX,2*CUR_HEIGHT         ;Push width and height
        PUSH    AX
        MOV     AX,2*CUR_WIDTH
        PUSH    AX
        PUSH    CS:Save_Area_y          ;Push x and y of destination
        MOV     AX,MIX_OFFSET
        PUSH    AX
        PUSH    CS:Save_Area_y          ;Push x and y of source
        MOV     AX,CUR_OFFSET
        PUSH    AX
        CALL    _BitBlt
        ADD     SP,12

        ; Mix AND & XOR masks into build area (this will work only if all of
        ; the save area is in the same segment!!!)
```

```
        MOV     CX,Arg_Curs_x           ;Fetch x
        AND     CX,CUR_WIDTH-1          ;Keep 'odd' bits
        ADD     CX,MIX_OFFSET           ;Add 'base x' of save area
        MOV     AX,Arg_Curs_y           ;Fetch y
        AND     AX,CUR_HEIGHT-1         ;Keep 'odd' bits
        ADD     AX,CS:Save_Area_y       ;Add 'base y' of save area
        MUL     CS:Video_Pitch          ;   multiply y by width in bytes
        ADD     AX,CX                   ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits

        MOV     DI,AX                   ;Save offset
        MOV     AL,DL                   ;Select page
        CALL    Select_Page
        MOV     ES,CS:Graf_Seg          ;Set both segments to video buffer
        MOV     DS,CS:Graf_Seg

        MOV     DL,CUR_HEIGHT           ;Initialize raster counter
        MOV     SI,CS:Save_Offset       ;Get pointer to AND & XOR masks
        MOV     BX,CS:Video_Pitch       ;Compute scan-to-scan increment
        SUB     BX,CUR_WIDTH

Mix_Lines:
        MOV     CX,CUR_WIDTH            ;Fetch cursor width
Mix_Bytes:
        LODSB                           ;Fetch next byte of AND mask
        MOV     AH,[DI]                 ;Fetch next byte of destination
        AND     AL,AH                   ;Combine mask with destination
        MOV     AH,[SI+CUR_WIDTH-1]     ;Fetch next byte of XOR mask
        XOR     AL,AH                   ;Combine with previous result
        STOSB                           ;Place result into destination
        LOOP    Mix_Bytes

        ADD     DI,BX                   ;Point to next scanline
        ADD     SI,BX                   ;Point to next scanline
        DEC     DL                      ;Check if all scanlines done
        JG      Mix_Lines               ;Go do next scanline if not done

        ; Use _BitBlt procedure to copy build area to screen (and erase old
        ; cursor with the new cursor block).

        MOV     AX,2*CUR_HEIGHT         ;Push width and height
        PUSH    AX
        MOV     AX,2*CUR_WIDTH
        PUSH    AX
        PUSH    CS:Last_Cursor_y        ;Push x and y of destination
        PUSH    CS:Last_Cursor_x
        PUSH    CS:Save_Area_y          ;Push x and y of source
        MOV     AX,MIX_OFFSET
        PUSH    AX
        CALL    _BitBlt
        ADD     SP,12

        ; Clean up and return

        POP     DS                      ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI

        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
Move_SW_Cursor ENDP

;*************************************************************************
;*                                                                       *
;* _Remove_Cursor                                                        *
;*      This procedure is used to remove the cursor from the screen     *
;*      and to restore the screen to its original appearance           *
;*                                                                       *
;*************************************************************************
```

```
Remove_SW_Cursor PROC NEAR
        PUSH    BP                      ;Standard high-level entry
        MOV     BP,SP

        PUSH    SI                      ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Use _BitBlt to restore area under the last cursor location

        MOV     AX,2*CUR_HEIGHT         ;Push width and height
        PUSH    AX
        MOV     AX,2*CUR_WIDTH
        PUSH    AX
        PUSH    CS:Last_Cursor_y        ;Push last position of cursor
        PUSH    CS:Last_Cursor_x
        PUSH    CS:Save_Area_y          ;Push x and y of destination
        MOV     AX,CUR_OFFSET
        PUSH    AX
        CALL    _BitBlt
        ADD     SP,12

        ; Clean up and return

        POP     DS                      ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI

        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
Remove_SW_Cursor ENDP

_TEXT   ENDS
        END
```

## Detection and Identification

The Cirrus BIOS contains a signature code (ASCII 'CL') at address C000:0006. To test for the presence of a Cirrus BIOS, code similar to the following can be used:

```
                ; Check of Cirrus based BIOS
                MOV             AX,0C000h               ;Fetch segment of ROM BIOS
                MOV             ES,AX
                CMP             WORD PTR ES:[6],'CL'    ;Is Cirrus signature present?
                JNE             Not_Cirrus_BIOS         ;...No, quit
Cirrus_BIOS_Found:
```

An alternate test can be used if the BIOS is not accessible, or if the BIOS does not conform to Cirrus recommendations. Extended registers addressed at 3C4h must first be enabled for writing by writing an unlocking password to the Extention Control register (index 6). To disable access to extended registers, a locking password must be written to the Extension Control register. All other values are ignored by this register. When the extended register is read back, bits D7 through D1 are returned as 0, and bit D0 returns the lock/unlock status (0 = locked, 1 = unlocked).

The unlock password value can be obtained by first clearing the Start Address register (CRTC index 0Ch) to zero, then reading the Identification register (CRTC index

1Fh). This code can be used to detect the presence of a Cirrus VGA chip. For chip set 510/520 this value is ECh, and for chip set 610/620 it is CAh. Video Seven boards that are based on Cirrus chips use a value of EAh. The lock password can be derived from the unlock password by reversing the nibbles (or rotating by 4).

To verify the presence of a Cirrus VGA chip, the following code can be used:

```
                ; Fetch address of CRT controller
                XOR         AX,AX                       ;Segment of BIOS data area
                MOV         ES,AX
                MOV         DX,ES:[463h]                ;Fetch CRTC address
                PUSH        DX                          ;Save for later
                ; Clear Start Address register in CRTC (index 0Ch)
                MOV         AL,0Ch                      ;Index of Start Address register
                OUT         DX,AL                       ;Select Start Address register
                INC         DX
                IN          AL,DX                       ;Get current value of register
                MOV         AH,AL                       ;Save to be restored later
                MOV         AL,0Ch
                PUSH        AX
                XOR         AL,AL                       ;Value for Start Address reg
                OUT         DX,AL                       ;Clear Start Address register
                DEC         DX
                ; Fetch Unlock Password
                MOV         AL,1Fh                      ;Index of Identification reg
                OUT         DX,AL                       ;Select ID registers
                INC         DX
                IN          AL,DX                       ;Read Unlock Password
                MOV         AH,AL                       ;Save for later
                ; Enable extended registers
                MOV         DX,3C4h                     ;Address of Sequencer
                MOV         AL,06h                      ;Index of Extension Control reg
                OUT         DX,AL                       ;Select Extension Control reg
                INC         DX
                MOV         AL,AH                       ;Fetch Unlock Password
                OUT         DX,AL                       ;Enable extended registers
                IN          AL,DX                       ;Read back Extension Reg
                CMP         AL,1                        ;Is it read back as 1?
                JNE         Not_Cirrus                  ;...No, canot be cirrus
                ; Disable extended registers
                MOV         AL,AH                       ;Fetch Unlock Password
                ROR         AL,1                        ;Compute Lock Password
                ROR         AL,1
                ROR         AL,1
                ROR         AL,1
                OUT         DX,AL                       ;Lock extended registers
                IN          AL,DX                       ;Read Extended Control reg
                OR          AL,AL                       ;Is it zero?
                JNE         Not_Cirrus                  ;...No, cannot be cirrus
Cirrus_Found:
                POP         AX                          ;Fetch original CRTC value
                POP         DX                          ;Fetch address of CRT
                OUT         DX,AX                       ;Restore registerC
```

# 13

# Chips and Technologies 82C452 Boca 1024VGA

# Introduction

Boca Research has developed 1024VGA, an enhanced VGA-compatible display adapter based on the Chips and Technologies 82C452 VGA chip. This chip has a number of advanced features, including hardware support for a graphics cursor and a memory paging mechanism that permits dual read-writable memory windows. It also supports emulation modes for compatibility with EGA, CGA, MDA and Hercules text and graphics modes. The board will support resolutions as high as 1024x768 with 16 colors or 640x480 with 256 colors.

BOCA 1024VGA includes a 16-bit bus interface and can be used in either 8- or 16-bit card slots. It includes 512K of onboard display memory. Only analog video output is supplied (TTL displays are not supported).

# New Display Modes

Table 13-1 lists the enhanced display modes that are supported by the 1024VGA. Any of these modes can be selected by issuing a BIOS Mode Select command.

**Table 13-1. Enhanced modes—1024VGA**

| Mode | Type | Resolution | Colors | Display Type |
|---|---|---|---|---|
| 60h | Text | 132 col x 25 rows | 16 | VGA |
| 61h | Text | 132 col x 50 rows | 16 | VGA |
| 6Ah | Graphics | 800 x 600 | 16 | SuperVGA |
| 72h | Graphics | 1024 x 768 | 16 | XL |
| 78h (1) | Graphics | 640 x 400 | 256 | VGA |
| 79h | Graphics | 640 x 480 | 256 | VGA |
| 7Ah (1) | Graphics | 720 x 540 | 256 | VGA |

Note: These modes are not documented in 1024VGA manual.

# Memory Organization

For all extended display modes of the 1024VGA, display memory organization is closely patterned after the organization used in standard IBM VGA modes.

The 1024VGA includes a powerful display memory paging mechanism that is needed in some display modes to make the entire display memory accessible to the processor. Display memory paging is described in detail later in this chapter.

## High Resolution Text Modes

These modes utilize memory maps that are similar to those used in standard text modes (modes 0,1,2,3 and 7), except that the number of characters per line, or lines per screen, is increased. Display memory is organized as shown in Figure 5-1 (see Chapter 5).

## Modes 6Ah, 72h - 800x600 and 1024x768 (16 Colors)

Display memory organization for modes 6Ah and 72h conforms closely to that of VGA mode 12h, except that the number of pixels in a scanline and the number of scanlines is increased. In mode 72, memory can either be addressed as one 128K page, or as two 64K pages. A detailed description and programming examples for this type of mode, using two 64K pages, are shown in Chapter 7.

## Mode 79h - 640x480 (256 Colors)

For this mode, display memory is organized similarly to VGA mode 13h. See Chapter 8 for a detailed description and programming examples.

# New Registers

Registers internal to the Chips and Technologies 82C452 VGA chip may be used to enable enhanced features of the board. Some of the added registers are not often used by programmers; we have included here a list of those registers which may be interesting or useful in typical drawing operations. Table 13-2 contains a list of new registers which are mentioned in this chapter.

**Table 13-2. New Registers**

| Address | Index | Description |
|---|---|---|
| 46E8h | | Setup Control Register for AT based boards |
| 94h | | Setup Control Register for Micro Channel boards |
| 103h | | Extended Enable Register (in Setup mode only) |
| 104h | | Global ID (in Setup mode only) |
| 3D4h/3B4h | 22h | CPU Data Latch or Color Compare from last read |
| | 24h | Attribute controller flip/flop |
| EXTENDED REGISTER BANK | | |
| 3D6h | 00 | Chips Version |
| | 01 | Dip Switch |
| | 02 | CPU Interface |

Table 13-2. New Registers *(continued)*

| Address | Index | Description |
|---|---|---|
| | 03 | ROM Interface |
| | 04 | Memory Mapping |
| | 05 | Sequencer Control |
| | 06 | DRAM Interface |
| | 08 | General purpose |
| | 09 | General purpose |
| | 0Ah | Cursor Page |
| | 0Bh | Memory Paging Register |
| | 0Ch | Start Address Top |
| | 0Dh | Auxiliary Offset |
| | 0Eh | Text Mode |
| | 10h | Memory Page 1 Base Address |
| | 11h | Memory Page 2 Base Address |
| | 20h | Sliding Unit Delay |
| | 21h | Sliding Hold A |
| | 22h | Sliding Hold B |
| | 23h | Sliding Hold C |
| | 24h | Sliding Hold D |
| | 27h | Force Sync State |
| | 28h | Video Interface |
| | 29h | External Sync Control |
| | 2Ah | Frame Interrupt Count |
| | 2Bh | Default Video |
| | 2Ch | Force Horizontal High |
| | 2Dh | Force Horizontal Low |
| | 2Eh | Force Vertical High |
| | 2Fh | Force Vertical Low |
| | 30h | Graphics Cursor Start Address High |
| | 31h | Graphics Cursor Start Address Low |
| | 32h | Graphics Cursor End Address |
| | 33h | Graphics Cursor X Position High |
| | 34h | Graphics Cursor X Position Low |
| | 35h | Graphics Cursor Y Position High |
| | 36h | Graphics Cursor Y Position Low |
| | 37h | Graphics Cursor Mode |
| | 38h | Graphics Cursor Plane Mask |
| | 39h | Graphics Cursor Color 0 |
| | 3Ah | Graphics Cursor Color 1 |

All of the new registers can be both written and read by the processor. Register bits that are marked as reserved must have their previous contents preserved when the reg-

ister is modified. The Setup Control register (I/O address 46E8h) and Extended Enable register (I/O address 103h) are used to enable access to the extended register bank (for details see the programming examples later in this chapter). Once access is enabled, extended registers are accessed via the standard VGA (index, data) pair mechanism, like so:

```
MOV  DX,3D6h        ;Fetch I/O address
MOV  AL,Index       ;Fetch index
OUT  DX,AL          ;Select register
INC  DX             ;Point to data address
MOV  AL,New_Data    ;Fetch data value
OUT  DX,AL          ;Write new data value
```

Care should be taken not to alter any registers in the extended register bank other than those that are described here; otherwise the display mode may be corrupted. To learn more about register access, see the programming examples in this chapter (in particular Listing 13-1).

## Setup Control Register (I/O Address 46E8h on AT, 94h on Micro Channel)

The Setup Control register serves just two purposes: to enable or disable the VGA adapter, or to enable access to the Extended Enable register which enables the enhanced features of the board.

D7-D5 - reserved
D4 - Setup Mode (1 = Setup mode, 0 = Normal mode)
D3 - VGA Enable (1 = VGA enabled, 0 = VGA disabled)
D2-D0 - reserved

Setup Mode enables access to the Extended Enable register. Care must be taken to disable setup mode immediately after accessing the Extended Enable register. If the board is left in setup mode, improper operation may result.

## Extended Enable Register (I/O Address 103h in Setup Mode)

D7 -Extended Registers Access Enable (1 = enabled)
D6 -Extended Registers Address Select
  0 = I/O address 3D6 and 3D7
  1 = I/O address 3B6 and 3B7
D5-D0 - reserved

Access Enable allows access to the extended register bank.
Address Select determines what address the extended register bank will be mapped to.

## Global ID Register (I/O Address 104h in Setup Mode)

D7 to D0 - Always read as A5h

This register is a read-only register which always reads back the value A5h to identify it as a Chips product.

## Extended Register Bank (I/O Address 3D6 or 3B6)

An index is written to this register to select which extended register will be accessed. After the extended register has been selected it is accessed via I/O address 3D7 or 3B7. The two most interesting groups of extended registers are the display memory paging registers and the cursor control registers.

The display memory paging mechanism of the Chips and Technologies 82C452 chip is one of the most flexible and powerful of any VGA chip. It permits two completely independent memory pages to be selected, each with read and write capability, with varying size and granularity. The page size is either 32K or 64K, and the granularity is as low as 4K (see "Display Memory Paging" in Chapter 5 for more details on granularity and page size).

Although this powerful paging scheme can be used to improve some drawing algorithms, the discussion and examples in this book assume 64K pages with 64K granularity for consistency.

### *Index 00h - CHIPS Version Register*

D7 to D4 -Chip type
    0 - 82C451
    1 - 82C452
    2 - Not used
    3 - 82C453
D0 to D3 -Silicon version

### *Index 0Bh - Memory Paging Register*

D7-D3 - Not used
D2 - CPU address divide by 4 (256 color addressing)
D1 - Dual Page Enable (1 = enabled)
D0 - 0 = Normal, 1 = Enable extended paging (256 color paging)

Dual Page Enable allows two pages of display memory to be opened simultaneously at two different host memory addresses. This is extremely useful when data must be moved from one page of display memory to another, which is frequently the case dur-

ing BITBLT operations. Memory Page Base Address registers, described below, define what section of display memory will be visible at each page. The Miscellaneous register of the Graphics Controller defines what host address each page will be mapped at (see Table 13-3).

## Index 10h - Memory Page 1 Base Address

D7-D6 - reserved
D5-D0 - Page Address

This register defines the base address of the first page of display memory; in other words, it defines what section of display memory will be visible to the host in page 1. In 256-color modes, the Memory Page 1 Base Address register contains address bits A14 through A19, and memory pages start on a 16K boundary. For 16-color modes, memory pages start on 4K boundary, and the contents of this register are added to address bits A12 through A19. This is illustrated in Figure 13-2 on page 317.

Memory page size is determined by the dual page enable bit, and by host window size as indicated in Table 13-3. To learn more about this register see the programming examples later in this chapter.

## Index 11h - Memory Page 2 Base Address

D7-D6 - reserved
D5-D0 - Page Address

If dual map mode is enabled, this register defines the base address of the second page of display memory. The contents of this register is similar to that of Memory Page 1 Base Address described in the previous section, but are used for display memory page 2. To learn more about this register see the programming examples in this chapter.

**Table 13-3. Display memory page addresses**

| 3CEh - Index 6 | | Page Size and Start Address | |
|---|---|---|---|
| **bits 3&2** | **Host Address Window** | **Page 1** | **Page 2** |
| 0 0 | A000:0 - BFFF:F | 64kB, A000:0 | 64kB, B000:0 |
| 0 1 | A000:0 - AFFF:F | 32kB, A000:0 | 32kB, A800:0 |
| 1 0 | B000:0 - B7FF:F | 64kB, B000:0 | disabled |
| 0 0 | B800:0 - BFFF:F | 32kB, B800:0 | disabled |

## 0Ch - Start Address Top

With more on board display memory than the standard VGA, the Start Address register of the CRT Controller is no longer sufficient to define a memory start address for screen refresh. This register defines the topmost address bits for the start address. See part one of the book for an explanation of the Start Address register.

D7-D2 - Reserved
D1-D0 - Bits A17, A16 in the CRTC Start Address register

## The Graphics Cursor

A hardware controlled graphics cursor can greatly simplify the task of cursor control in graphical environments such as Microsoft Windows or GEM, where the cursor is usually represented by a graphic icon such as an arrow or a crosshair. This cursor operates as an overlay on the screen. Unlike a software controlled cursor, on-screen display memory is not altered by the cursor. The 82C452 chip provides 32-bit wide cursor support for all modes supported by the chip. This includes support for 16-color and 256-color graphics modes, as well as for all text modes. Even in text modes that use a 9-pixel wide character cell, cursor position can be controlled to single pixel resolution.

Twelve registers are required to control the operation of the graphics cursor. They are logically divided as follows:

- **Cursor Mode Control** (Index 37h) - Sets the operating mode of the graphics cursor.
- **Cursor Address** (Index 0Ah, 30h, 31h, 32h) - The cursor pattern displayed on the screen is defined by a data pattern, or bitmap, stored in off-screen display memory. These registers define the start and end address for that pattern.
- **Cursor Position** (Index 33h through 36h) - These registers control the position of the graphics cursor on the screen. X and Y coordinates are used to position the cursor.
- **Cursor Color** (Index 38h, 39h and 3Ah) - Cursor shape is defined by a pattern in two monochrome bit maps stored in off-screen memory. This pattern may be color expanded to a foreground color and a background color when it is displayed. Cursor color registers define the foreground and background colors.
- **Cursor definition** and **control** is illustrated in Figure 13-1. To learn more about programming the cursor registers see the programming examples at the end of this chapter.

### Index 37h - Graphics Cursor Mode

D7-D5 - unused
D4 - Cursor Blink Rate (0 = 8 frames, 1 = 16 frames)
D3 - Cursor Blink Enable (1 = enabled)
D2 - Horizontal Zoom
  1 = zoom cursor to 64 pixels wide
  (Cursor is normally 32 pixels wide)
D1 - Cursor Status Enable
D0 - Cursor Enable (1 = enabled)

### Index 0Ah - Graphics Cursor Page

D7-D2 - Reserved
D1-D0 - Top two bits of the start address in display memory for the graphics cursor pattern data

This register, together with the Graphics Cursor Start Address High (index 30h), defines the 20-bit address of the cursor pattern in display memory. This register is used only on boards containing 1024K of display memory.

### Index 30h - Graphics Cursor Start Address High

This register, together with the Graphics Cursor Start Address Low register (index 31h) and the Graphics Cursor Page register (index 0Ah), defines the start address of the cursor pattern in display memory. In 256-color modes, this start address has a granularity of 16 bytes. In 16-color modes, it has a granularity of 4 bytes.

### Index 31h - Graphics Cursor Start Address Low

This register, together with the Graphics Cursor Start Address High register (index 30h) and the Graphics Cursor Page register (index 0Ah), defines the start address of the cursor pattern in display memory. In 256-color modes, this start address has a granularity of 16 bytes. In 16-color modes, it has a granularity of 4 bytes.

### Index 32h - Graphics Cursor End Address

This register, together with the other graphics cursor address registers, defines the end address of the cursor pattern in display memory. In 256-color modes, this address has a granularity of 16 bytes. In 16-color modes, it has a granularity of 4 bytes.

## *Index 33h - Graphics Cursor X Position High and Index 34h - Graphics Cursor X Position Low*

These two registers contain the X (horizontal) coordinate that is used to specify the location of the hardware graphics cursor on the display screen. The X coordinate is a twelve-bit value.

## *Index 35h - Graphics Cursor Y Position High and Index 36h - Graphics Cursor Y Position Low*

These two registers contain the Y (vertical) coordinate that is used to specify the location of the hardware graphics cursor on the display screen. The Y coordinate is a twelve-bit value.

## *Cursor Color Registers*

The graphics cursor pattern is stored in off-screen memory as two monochrome bitmaps which are expanded to two colors when the cursor is displayed. COLOR1 defines the color that will be produced by a code of 11 in the two bitmaps. COLOR0 defines the color that will be produced by a code of 10 in the two bitmaps. The Plane Mask can modify these colors by excluding certain color planes.

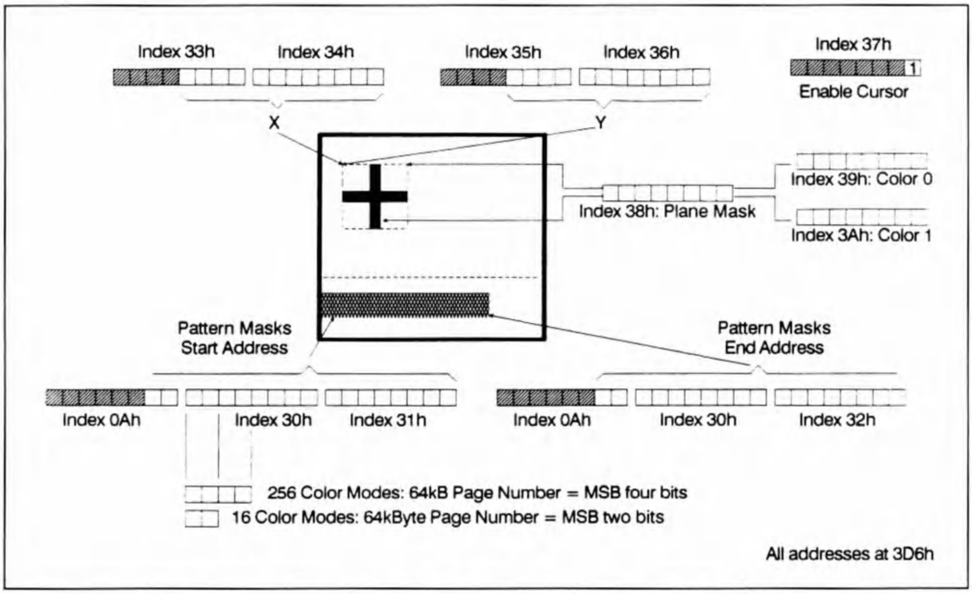


Figure 13-1. Graphics Cursor Controls

### Index 38h - Graphics Cursor Plane Mask

A value of 0 in any bit position in this register disables the graphics cursor from affecting the corresponding color plane.

### Index 39h - Graphics Cursor Color 0

This register defines the value of color 0 (background color) to be used when the graphics cursor is color expanded.

### Index 3Ah - Graphics Cursor Color 1

This register defines the value of color 1 (foreground color) to be used when the graphics cursor is color expanded.

# The BIOS

The VGA1024 does not have any documented BIOS services beyond those provided by the standard VGA. The Chips and Technologies BIOS supports Extended VGA Control via BIOS function 5Fh.

## Function 5Fh - Subfunction 00h: Return 82C45x Information

**Input:**

AH = 5Fh
AL = 00h

**Output:**

AL = 5Fh
BL = Chip type and revision number
  D7-D4 = Chip type
    0: 82C451
    1: 82C452
    3: 82C453
BH = Display memory size
  0: 256K
  1: 512K
  2: 1024K
CX = Miscellaneous information
  D0 = DAC size (0: 6-bit, 1: 8-bit)
  D1 = Environment (0: AT, 1: MCA)

D2 = Extended text modes supported by BIOS
D3 = Reserved
D4 = Extended graphics mode supported by BIOS
D5 = Reserved
D6 = Graphics cursor supported by BIOS
D7 = Anti-alias font supported by BIOS
D8 = Preprogrammed emulation supported by BIOS
D9 = Auto emulation supported by BIOS
D10 = Variable mode set at cold boot supported by BIOS
D11 = Variable mode set at warm boot supported by BIOS
D12 = Emulation set at cold boot supported by BIOS
D13 = Emulation set at warm boot supported by BIOS
D14-D15 = Reserved

## Function 5Fh - Subfunction 01h: Preprogrammed Emulation Control

**Input:**

AH = 5Fh
AL = 01h
BL = Emulation control code
2: Enable and lock CGA emulation
3: Enable and lock MDA emulation
4: Enable and lock Hercules emulation
5: Enable and lock EGA emulation
6: Disable emulation (normal VGA operation)

**Output:**

AL = 5Fh
AH = 1 if successful, 0 if failed

## Function 5Fh - Subfunction 02h: Auto-emulation Control

**Input:**

AH = 5Fh
AL = 02h
BL = Emulation control code
0: Enable emulation
1: Disable auto-emulation

**Output:**

AL = 5Fh
AH = 1 if successful, 0 if failed

## Function 5Fh - Subfunction 03h: Set Power-on Video Conditions

**Input:**

AH = 5Fh
AL = 03h
BL = 00h
CL = Display mode
CH = Scanlines (0: 200, 1: 350, 2: 400)
BL = 01h
CL = Emulation mode (See subfunction 1 above)
CH = Permanence (0: Reset after next boot,
1: Keep until changed)

**Output:**

AL = 5Fh
AH = 1 if successful, 0 if failed

## Function 5Fh - Subfunction 90h: Enhanced Save/Restore Video State Buffer Size

**Input:**

AH = 5Fh
AL = 90h
CX = Mask of states to save
D0 - Hardware
D1 - BIOS data area
D2 - DAC
D15 - Type (0: all, 1: 82C45x specific)

**Output:**

AL = 5Fh
BX = Number of 64-byte blocks necessary

## Function 5Fh - Subfunction 91h: Save Video State

**Input:**

AH = 5Fh
AL = 91h
CX = Mask of states to save
D0 - Hardware
D1 - BIOS data area
D2 - DAC
D15 - Type (0: all, 1: 82C45x specific)
ES:BX = Buffer address

**Output:**

AL = 5Fh

## Function 5Fh - Subfunction 92h: Restore Video State

**Input:**

AH = 5Fh
AL = 92h
CX = Mask of states to save
D0 - Hardware
D1 - BIOS data area
D2 - DAC
D15 - Type (0: all, 1: 82C45x specific)
ES:BX = Buffer address

**Output:**

AL = 5Fh

# Programming Examples

## Accessing Extended Registers

When writing software to take advantage of the extended features of the 1024VGA, it is important to note that the extended register bank must first be enabled before it can be accessed. Enabling and disabling of the extended register bank is performed by placing the VGA chip in setup mode, then modifying the Extended Enable register, with code similar to the following (for AT based systems):

```
CLI
; Place VGA in SETUP mode
MOV          DX,46E8h           ;address the setup control register
IN           AL,DX
OR           AL,10h
OUT          DX,AL              ;place it in setup mode
; Enable extended register bank
MOV          DX,103h            ;address the extended enable register
IN           AL,DX
OR           AL,80h
OUT          DX,AL              ;enable extended register bank
; Place VGA in NORMAL mode
MOV          DX,46E8h           ;address the setup control register
IN           AL,DX
AND          AL,0EFh
OUT          DX,AL              ;disable setup mode
STI
```

To learn more about programming the Extended register see Listing 13-1 on the following page.

## Display Memory Paging

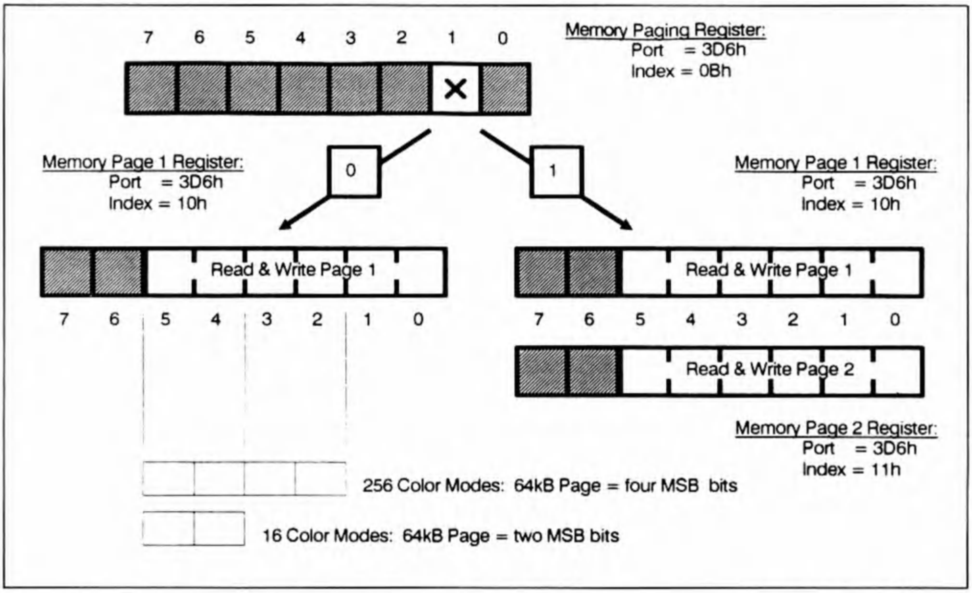


Figure 13-2. Memory Paging registers

The 1024VGA contains a display memory paging mechanism that maps selected portions of the display memory to the processor. The operation of display memory paging is very similar to the paging mechanism used for expanded memory boards (also called EMS or LIM memory). A 64K logical page of VGA RAM (a chunk of display memory) is mapped into the PC host address space in the normal VGA display memory

address space. Extended I/O registers at 3D6 or 3B6, Index 0Bh (Memory Paging register), Index 10h (Page 1 Base Address), and Index 11h (Page 2 Base Address) are used to select memory pages. Figure 13-2 illustrates the structure of the Memory Paging registers. To learn more about paging, see Chapter 5.

Either one or two display memory pages may be enabled. Unlike many other VGA products, both memory pages are simultaneously readable and writable. This can be very useful when transferring data from one part of display memory to another (BITBLT).

Listing 13-1 illustrates how the paging registers are used. The extended register bank is enabled in the procedure _Select_Graphics. To easily interface with the common drawing routines, the paging routines in listing 13-1 do not take full advantage of the Chips and Technologies chip capabilities. Select_Page assumes that a 64K page has been requested with a granularity of 64K. Select_Read_Page and Select_Write_Page assume that a 32K page has been requested with a granularity of 32K, and that the read page is in page 1, addressed by DS at A000, and the write page is in page 2 addressed by ES at A800.

It should be noted that with the Boca Research BIOS, display memory beyond 256K is disabled in standard VGA modes.

**Listing 13-1. File: CTI\SELECT.ASM**

```
;*************************************************************************
;* File:        SELECT.ASM                                              *
;* Description: This module contains procedures to select mode and to   *
;*              select pages.  It also initializes global variables     *
;*              according to the values in the MODE.INC include file.   *
;* Entry Points:                                                        *
;*              _Select_Graphics    - Select a graphics mode            *
;*              _Select_Text        - Set VGA adapter into text mode    *
;*              _Select_Page        - Select 64k page                   *
;*              _Select_Read_Page   - Select 32k page A                 *
;*              _Select_Write_Page  - Select 32k page B                 *
;* Uses:                                                                *
;*              MODE.INC            - Mode dependent constants          *
;*              Following are modes and paths for Boca 1024 board:      *
;*      |----- 256 colors ------|  |-- 16 colors --|  4 colors 2 colors *
;*      640x400  640x480  800x600  800x600  1024x768  1024x768 1024x768 *
;* Mode:  78h      79h      N/A      6Ah      72h       N/A      N/A    *
;* Path: 256COL  256COL     N/A     16COL    16COL      N/A      N/A    *
;*************************************************************************

     INCLUDE VGA.INC
     INCLUDE MODE.INC                   ;Mode dependent constants

     PUBLIC    _Select_Graphics
     PUBLIC    _Select_Text
     PUBLIC    _Select_Page
     PUBLIC    _Select_Read_Page
     PUBLIC    _Select_Write_Page

     PUBLIC    Select_Page
     PUBLIC    Select_Read_Page
     PUBLIC    Select_Write_Page
     PUBLIC    Enable_Dual_Page
     PUBLIC    Disable_Dual_Page
```

```
     PUBLIC    Graf_Seg
     PUBLIC    Video_Height
     PUBLIC    Video_Width
     PUBLIC    Video_Pitch
     PUBLIC    Video_Pages
     PUBLIC    Video_Colors
     PUBLIC    Ras_Buffer
     PUBLIC    Two_Pages

     PUBLIC    Last_Byte
;-----------------------------------------------------------------------
; Data segment variables
;-----------------------------------------------------------------------

;_DATA     SEGMENT WORD PUBLIC 'DATA'
;_DATA     ENDS

;-----------------------------------------------------------------------
; Constant definitions
;-----------------------------------------------------------------------

PAGE_CTL_REG        EQU  0Bh              ;Index for page control register
PAGE1_REG           EQU  10h              ;Index for page 1 select
PAGE2_REG           EQU  11h              ;Index for page 2 select
PAGE_MASK           EQU  0C0h             ;Page mask
EXTENDED_PORT       EQU  3D6h             ;Address of extended registers
CPU_PAGING_REG           EQU   0Bh
EXTEND_ENABLE_PORT       EQU   0103h
SETUP_PORT               EQU   46E8h

;-----------------------------------------------------------------------
; Code segment variables
;-----------------------------------------------------------------------

_TEXT     SEGMENT BYTE PUBLIC 'CODE'

Graf_Seg            DW   0A000h           ;Graphics segment addresse
                    DW   0A800h           ;Second page address
OffScreen_Seg       DW   0A000h           ;First byte beyond visible screen
Video_Pitch         DW   SCREEN_PITCH     ;Number of bytes in one raster
Video_Height        DW   SCREEN_HEIGHT    ;Number of rasters
Video_Width         DW   SCREEN_WIDTH     ;Number of pixels in a raster
Video_Pages         DW   SCREEN_PAGES     ;Number of pages in the screen
Video_Colors        DW   SCREEN_COLORS    ;Number of colors in this mode
Ras_Buffer          DB   1024 DUP (0)     ;Working buffer
R_Page              DB   0FFh             ;Most recently selected page
W_Page              DB   0FFh
RW_Page             DB   0FFh
Two_Pages           DB   CAN_DO_RW        ;Indicate separate R & W capability

;***********************************************************************
;*                                                                     *
;* _Select_Graphics(HorizPtr, VertPtr, ColorsPtr)                      *
;*     Initialize VGA adapter to graphics                              *
;*                                                                     *
;* Entry:                                                              *
;*     None                                                            *
;*                                                                     *
;* Returns:                                                            *
;*     VertPtr   - Vertical resolution                                 *
;*     HorizPtr  - Horizontal resolution                               *
;*     ColorsPtr - Number of supported colors                          *
;*                                                                     *
;***********************************************************************

Arg_HorizPtr   EQU  WORD PTR [BP+4]     ;Formal parameters
Arg_VertPtr    EQU  WORD PTR [BP+6]     ;Formal parameters
Arg_ColorsPtr  EQU  WORD PTR [BP+8]     ;Formal parameters
```

```
_Select_Graphics PROC NEAR
        PUSH    BP                      ;Standard C entry point
        MOV     BP,SP

        PUSH    DI                      ;Preserve segment registers
        PUSH    SI
        PUSH    DS
        PUSH    ES

        ; Select graphics mode

        MOV     AX,GRAPHICS_MODE        ;Select graphics mode
        INT     10h

        ;Make sure the extended register bank is enabled

        MOV     DX,SETUP_PORT
        MOV     AL,18h
        OUT     DX,AL                   ;Enable setup

        MOV     DX,EXTEND_ENABLE_PORT
        MOV     AL,80h
        OUT     DX,AL                   ;Enable extended registers

        MOV     DX,SETUP_PORT
        MOV     AL,8
        OUT     DX,AL                   ;Disable setup

        ; Enable access to memory beyond 256k (VGA modes do not do this).
        ; (This can be usefull when data needs to be stored in offscreen
        ; display memory.)

        MOV     DX,EXTENDED_PORT        ;Point to extended register bank
        MOV     AL,CPU_PAGING_REG       ;Index of CPU paging
        OUT     DX,AL                   ;Select register
        INC     DX
        IN      AL,DX                   ;Read previous value
        OR      AL,01h                  ;Set mapping mode bit
        OUT     DX,AL                   ;Enable additional memory

        ; Reset 'last selected page'

        MOV     AL,0FFh                 ;Use 'non-existent' page number
        MOV     CS:R_Page,AL            ;Set currently selected page
        MOV     CS:W_Page,AL
        MOV     CS:RW_Page,AL

        ; Set return parameters

        MOV     SI,Arg_VertPtr          ;Fetch pointer to vertical resolution
        MOV     WORD PTR [SI],SCREEN_HEIGHT   ;Set vertical resolution
        MOV     SI,Arg_HorizPtr         ;Fetch pointer to horizontal resolution
        MOV     WORD PTR [SI],SCREEN_WIDTH    ;Set horizontal resolution
        MOV     SI,Arg_ColorsPtr        ;Fetch pointer to number of colors
        MOV     WORD PTR [SI],SCREEN_COLORS   ;Set number of colors

        ; Clean up and return to caller

        POP     ES                      ;Restore segment registers
        POP     DS
        POP     SI
        POP     DI

        MOV     SP,BP                   ;Standard C exit point
        POP     BP
        RET
_Select_Graphics ENDP
```

```
;********************************************************************
;                                                                   *
; Select_Page                                                       *
; Entry:                                                            *
;       AL - Page number                                            *
;                                                                   *
;********************************************************************

Select_Page     PROC NEAR
        CMP     AL,CS:RW_Page           ;Check if already selected
        JNE     SP_Go
        RET
SP_Go:
        PUSH    AX
        PUSH    DX
        AND     AL,7                    ;Force page number into range
        MOV     CS:RW_Page,AL           ;Save as most recent RW page
        MOV     CS:R_Page,0FFh          ;Invalidate R and W pages
        MOV     CS:W_Page,0FFh

IFE (SCREEN_COLORS - 256)
        SHL     AL,1                    ;for 256 color modes,
        SHL     AL,1                    ;convert 64KB page # to 4 KB page #
ELSE
        SHL     AL,1                    ;for 16 color modes,
        SHL     AL,1                    ;convert 64KB page # to 16 KB page #
        SHL     AL,1
        SHL     AL,1
ENDIF
        MOV     AH,AL                   ;Copy page number into AH
        MOV     DX,EXTENDED_PORT        ;Fetch extended register address
        MOV     AL,PAGE1_REG            ;Fetch page select index
        OUT     DX,AL                   ;Select page select register
        INC     DX

        IN      AL,DX                   ;Read current value of page select reg
        AND     AL,PAGE_MASK            ;Clear previous page setting
        OR      AL,AH                   ;Combine with new page selection
        OUT     DX,AL                   ;Select new page
        POP     DX
        POP     AX
        RET
Select_Page     ENDP

;********************************************************************
;                                                                   *
; Select_Read_Page                                                  *
;       Assumes that caller uses 32kByte page at DS for first page. *
; Entry:                                                            *
;       AL - Page number                                            *
;                                                                   *
;********************************************************************

Select_Read_Page PROC NEAR
        CMP     AL,CS:R_Page            ;Check if already selected
        JNE     SR_Go
        RET
SR_Go:
        PUSH    AX
        PUSH    DX
        AND     AL,0Fh                  ;Force page number into range
        MOV     CS:R_Page,AL            ;Save as most recent Read page
        MOV     CS:RW_Page,0FFh         ;Invalidate most recent RW page

IFE (SCREEN_COLORS - 256)
        SHL     AL,1                    ;for 256 color modes,
                                        ;convert 32KB page # to 4 KB page #
ELSE
        SHL     AL,1                    ;for 16 color modes,
        SHL     AL,1                    ;convert 32KB page # to 16 KB page #
        SHL     AL,1
```

```
ENDIF
        MOV     AH,AL                   ;Copy page number into AH
        MOV     DX,EXTENDED_PORT        ;Fetch extended register address
        MOV     AL,PAGE1_REG            ;Fetch page select index
        OUT     DX,AL                   ;Select page select register
        INC     DX

        IN      AL,DX                   ;Read current value of page select reg
        AND     AL,PAGE_MASK            ;Clear previous page setting
        OR      AL,AH                   ;Combine with new page selection
        OUT     DX,AL                   ;Select new page
        POP     DX
        POP     AX
        RET
Select_Read_Page ENDP

;*************************************************************************
;                                                                        *
; Select_Write_Page                                                      *
;       Assumes that called uses 32 kByte page at ES for second page.    *
;                                                                        *
; Entry:                                                                 *
;       AL - Page number                                                 * ;
*
;*************************************************************************

Select_Write_Page PROC NEAR
        CMP     AL,CS:W_Page            ;Check if already selected
        JNE     SW_Go
        RET
SW_Go:
        PUSH    AX
        PUSH    DX
        AND     AL,0Fh                  ;Force page number into range
        MOV     CS:W_Page,AL            ;Save as most recent Write page
        MOV     CS:RW_Page,0FFh         ;Invalidate RW page

IFE (SCREEN_COLORS - 256)
        SHL     AL,1                    ;for 256 color modes,
                                        ;convert 32KB page # to 4 KB page #
ELSE
        SHL     AL,1                    ;for 16 color modes,
        SHL     AL,1                    ;convert 32KB page # to 16 KB page #
        SHL     AL,1
ENDIF
        MOV     AH,AL                   ;Copy page number into AH
        MOV     DX,EXTENDED_PORT        ;Fetch extended register address
        MOV     AL,PAGE2_REG            ;Fetch page select index
        OUT     DX,AL                   ;Select page select register
        INC     DX

        IN      AL,DX                   ;Read current value of page select reg
        AND     AL,PAGE_MASK            ;Clear previous page setting
        OR      AL,AH                   ;Combine with new page selection
        OUT     DX,AL                   ;Select new page
        POP     DX
        POP     AX
        RET
Select_Write_Page ENDP

;*************************************************************************
;                                                                        *
; _Select_Page(PageNumber)                                               *
; _Select_Read_Page(PageNumber)                                          *
; _Select_Write_Page(PageNumber)                                         *
;       Entry points for high level languages                            *
; Entry:                                                                 *
;       PageNumber - Page number                                         *
;                                                                        *
;*************************************************************************
```

```
Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Page   PROC NEAR
        PUSH   BP                       ;Setup frame pointer
        MOV    SP,BP
        MOV    AL,Arg_PageNumber        ;Fetch argument
        POP    BP                       ;Restore BP
        JMP    Select_Page
_Select_Page   ENDP

_Select_Read_Page   PROC NEAR
        PUSH   BP                       ;Setup frame pointer
        MOV    SP,BP
        MOV    AL,Arg_PageNumber        ;Fetch argument
        POP    BP                       ;Restore BP
        JMP  Select_Read_Page
_Select_Read_Page   ENDP

_Select_Write_Page  PROC NEAR
        PUSH   BP                       ;Setup frame pointer
        MOV    SP,BP
        MOV    AL,Arg_PageNumber        ;Fetch argument
        POP    BP                       ;Restore BP
        JMP    Select_Write_Page
_Select_Write_Page  ENDP

;************************************************************************
;*                                                                      *
;* _Select_Text                                                         *
;*      Set VGA adapter to text mode                                    *
;*                                                                      *
;************************************************************************

_Select_Text   PROC NEAR
        MOV    AX,TEXT_MODE             ;Select mode 3
        INT    10h                      ;Use BIOS to reset mode
        RET
_Select_Text   ENDP

;************************************************************************
;*                                                                      *
;* Enable_Dual_Page                                                     *
;* Disable_Dual_Page                                                    *
;*      Enable and disable dual page paging.                            *
;*                                                                      *
;************************************************************************

Enable_Dual_Page    PROC NEAR
        MOV    DS,CS:GRAF_SEG[0]        ;Set DS to first page
        MOV    ES,CS:Graf_Seg[2]        ;Set ES to second page
        MOV    DX,EXTENDED_PORT         ;Address of extended registers
        MOV    AL,PAGE_CTL_REG          ;Index of page control register
        OUT    DX,AL                    ;Select page control register
        INC    DX
        IN     AL,DX                    ;Read previous value
        OR     AL,02h                   ;Set dual page bit
        OUT    DX,AL                    ;Enable dual page paging
        RET
Enable_Dual_Page    ENDP

Disable_Dual_Page   PROC NEAR
        MOV    DX,EXTENDED_PORT         ;Address of extended registers
        MOV    AL,PAGE_CTL_REG          ;Index of page control register
        OUT    DX,AL                    ;Select page control register
        INC    DX
        IN     AL,DX                    ;Read previous value
        AND    AL,NOT 02h               ;Clear dual page bit
        OUT    DX,AL                    ;Disable dual page paging
        RET
Disable_Dual_Page   ENDP

Last_Byte:
_Text   ENDS
        END
```

## Graphics Cursors

The 1024VGA includes hardware support for a graphics cursor that can significantly reduce the processor overhead required for cursor control. The hardware cursor of the Chips and Technologies 82C452 VGA chip can be used with bit addressability in any mode, including text modes. Figure 13-1 illustrates the operation of the hardware graphics cursor. Twelve registers in the extended register bank are involved in the definition and control of the graphics cursor.

Hardware cursors operate differently than software cursors. Since the cursor is drawn as an overlay on the screen, there is never any need to save background data in the cursor area. The cursor is defined by two monochrome bitmaps, similar to the AND and XOR masks used for software cursors (for more on software cursors, see our previous text, *Programmer's Guide to the EGA/VGA*). Table 13-3 shows the colors defined by the cursor masks, and their correspondence to AND and XOR masks.

**Table 13-3. Hardware cursor masks**

| Chips Masks | | Conventional Masks | | |
|---|---|---|---|---|
| **Mask A** | **Mask B** | **AND** | **XOR** | **Resulting Cursor Color** |
| 0 | 0 | 1 | 0 | Display unmodified background data |
| 0 | 1 | 0 | 1 | Display foreground (color value from color reg 1) |
| 1 | 0 | 1 | 1 | Display inverted background data |
| 1 | 1 | 0 | 0 | Display background (color value from color reg 0) |

Note: To convert from XOR & AND to A & B masks, the following formulas can be used:

A = NOT (AND_Mask XOR XOR_Mask)
B = NOT AND_Mask

Cursor pattern data must be loaded into off-screen display memory in a scrambled format, depending on display mode. Figure 13-3 on page 337 shows cursor pattern locations for 16-color modes. Each row of cursor, for each mask, is defined by four bytes of pattern (32 bits for each 32 pixel row of the cursor). Each byte defines 8 pixels, with the most significant bit corresponding to the left most pixel. Bytes for mask A are in planes 0 and 1, and mask B is in planes 2 and 3. Each byte is labeled as (column, row) to indicate which byte in the cursor it controls. Table 13-3 shows how the 2 bits from masks A and B determine the color of each pixel.

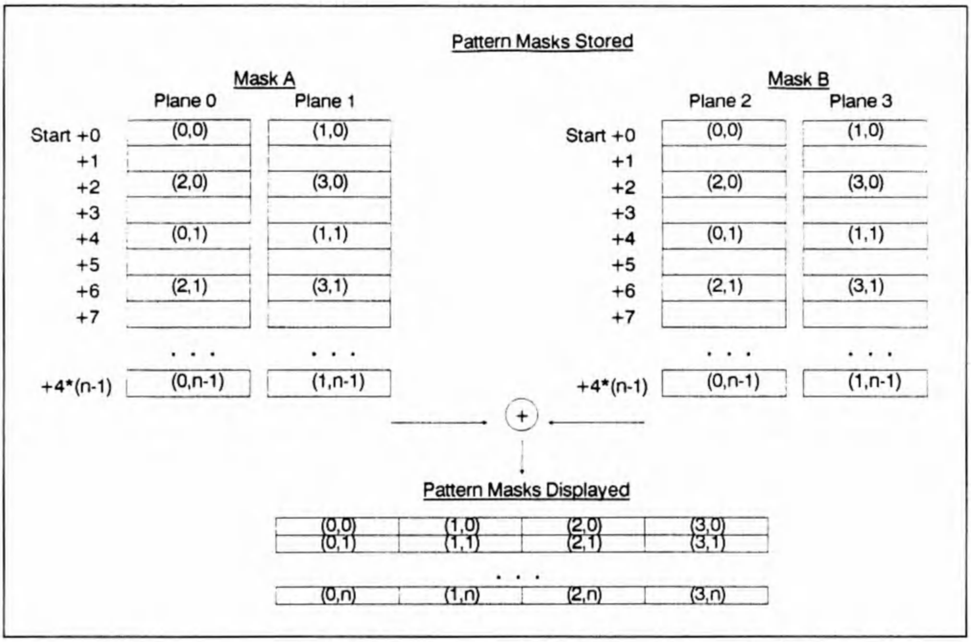


**Figure 13-3. Cursor masks in 16-color modes**

Figure 13-4 shows cursor pattern locations for 256-color modes. Each row of cursor is defined by two words of pattern in each mask (32 bits for 32 pixels of the cursor). Each word defines 16 pixels, with the most significant bit corresponding to the left most pixel. Words for mask A are stored at addresses which are multiples of 8, and words for mask A are followed by words for mask B. Each byte is labeled as (column, row) to indicate which byte in the cursor it controls. Table 13-3 shows how the 2 bits from masks A and B determine the color of each pixel.

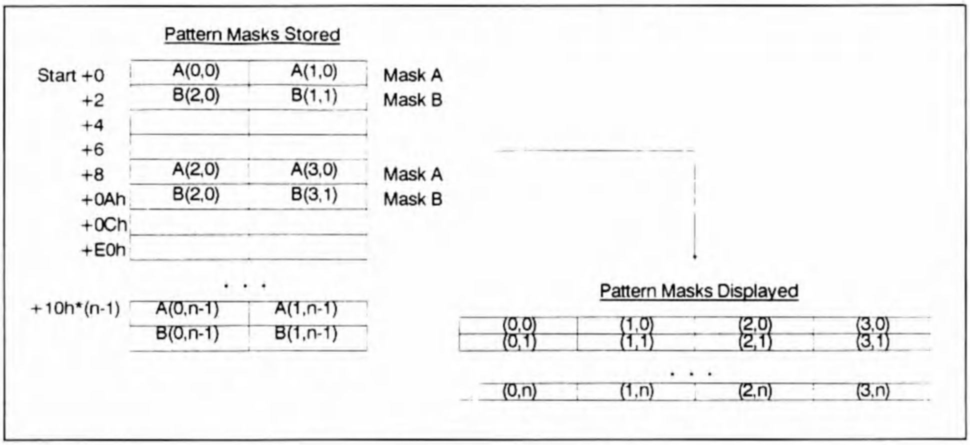


**Figure 13-4. Cursor masks in 256-color modes**

Listing 13-2 illustrates how to define cursor shape and how to move the cursor around the screen. Three procedures are provided:

**Set_Cursor** is used to convert standard AND and XOR masks to masks A and B, to define the color and shape of the cursor. **Move_Cursor** is used to determine where the cursor is displayed. **Remove_Cursor** disables the cursor display.

**Listing 13-2. File: CTI\HWCURSOR.ASM**

```
;**********************************************************************
;*                                                                    *
;* File:        HWCURSOR.ASM                                          *
;* Description: This module contains procedures to demonstrate use of a *
;*              hardware cursor.  It defines cursor shape, moves      *
;*              cursor around the screen, and removes cursor.         *
;*              This module does not work in mode 13h.                *
;*                                                                    *
;* Entry Points:                                                      *
;*              _Set_Cursor                                           *
;*              _Move_Cursor                                          *
;*              _Remove_Cursor                                        *
;*                                                                    *
;* Uses:                                                              *
;*              _Select_Page                                          *
;*              _Graf_Seg                                             *
;*              _Video_Height                                         *
;*              _Video_Pitch                                          *
;*                                                                    *
;**********************************************************************

        INCLUDE  VGA.INC
        INCLUDE  MODE.INC                ;Mode dependent constants

        EXTRN    Video_Pitch:WORD
        EXTRN    Video_Height:WORD
        EXTRN    Video_Colors:WORD
        EXTRN    Select_Page:NEAR
        EXTRN    Graf_Seg:WORD
        EXTRN    Video_Pages:BYTE
```

```
        PUBLIC   _Set_Cursor
        PUBLIC   _Move_Cursor
        PUBLIC   _Remove_Cursor

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

;-----------------------------------------------------------------------
; Common cursor register definitions
;-----------------------------------------------------------------------

EXTENDED_PORT       EQU  03D6H          ;extended features of CTI 452
CPU_PAGING_REG      EQU  0Bh
GCUR_XLO_REG        EQU  34h
GCUR_XHI_REG        EQU  33h
GCUR_YLO_REG        EQU  36h
GCUR_YHI_REG        EQU  35h
GCUR_MODE           EQU  37h
CURS_ADDR_HI_REG    EQU  0Ah
CURS_ADDR_MID_REG   EQU  30h
CURS_ADDR_LOW_REG   EQU  31h
CURS_ADDR_END_REG   EQU  32h
CURS_FG_REG         EQU  39h
CURS_BG_REG         EQU  3Ah
CURS_MASK_REG       EQU  38h

CURS_DX             EQU  32
CURS_DY             EQU  32

;***********************************************************************
;*                                                                     *
;* _Set_Cursor(AND_Mask, XOR_Mask, FG_Color, BG_Color)                 *
;*      This procedure will save cursor pattern in the off-screen      *
;*      memory according to the CTI schema.  Pattern is stored at last *
;*      512 bytes of last page (assumes 512kBytes of display memory)   *
;*                                                                     *
;* Entry:                                                              *
;*      AND_Mask - 4x32 bytes of inverted pattern A                    *
;*      XOR_Mask - 4x32 bytes of pattern B                             *
;*      BG_Color - Foreground color                                    *
;*      FG_Color - Background color                                    *
;*                                                                     *
;***********************************************************************

Arg_AND_Mask    EQU  WORD PTR [BP+4]     ;Formal parameters
Arg_XOR_Mask    EQU  WORD PTR [BP+6]
Arg_BG_Color    EQU  BYTE PTR [BP+8]
Arg_FG_Color    EQU  BYTE PTR [BP+10]

_Set_Cursor     PROC NEAR
        PUSH    BP                       ;Standard high-level entry
        MOV     BP,SP
        SUB     SP,2

        PUSH    SI                       ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Initialize pattern colors

        MOV     DX,EXTENDED_PORT         ;Point to extended register bank
        MOV     AL,CURS_FG_REG           ;Index of foreground color reg
        MOV     AH,Arg_FG_Color          ;Foreground color
        OUT     DX,AX                    ;Select foreground pattern color
        MOV     AL,CURS_BG_REG           ;Index of background color reg
        MOV     AH,Arg_BG_Color          ;Foreground color
        OUT     DX,AX                    ;Select background pattern color
        MOV     AL,CURS_MASK_REG         ;Index of color mask register
        MOV     AH,0FFh                  ;Enable all 8 bits for color
        OUT     DX,AX                    ;Select pattern color mask
```

```
        ;---------------------------------------------------------------
        ; Copy cursor masks for 256 color modes (take advantage of the fact
        ; that byte 4xN+P corresponds to byte N in plane P, for P=0,1,2 or 3)
        ;---------------------------------------------------------------

        CMP     CS:Video_Colors,256       ;Is this planar mode?
        JNE     SC_Do_Planar              ;No, go do planar mode

        ;Initialize Pattern Start Address to 0FE00 in last page
        ; Curs addr regs = 0FE0 + Page SHL 12, Page:Offset=Page:FE00

        MOV     AL,CURS_ADDR_MID_REG      ;Index of cursor address mid
        MOV     AH,CS:Video_Pages         ;Fetch number of visible pages
        DEC     AH                        ;Convert to page number
        ROR     AH,1                      ;Move page number to bits 4-7
        ROR     AH,1
        ROR     AH,1
        ROR     AH,1
        ADD     AH,0Fh                    ;Select last page
        OUT     DX,AX
        MOV     AL,CURS_ADDR_LOW_REG      ;Index of cursor address low
        MOV     AH,0E0h                   ;Select last page, last 512 bytes
        OUT     DX,AX

        ; Initialize Pattern End Address

        MOV     AL,CURS_ADDR_END_REG      ;Index of cursor address end
        ADD     AH,01Fh                   ;Set end 32 lines after start
        OUT     DX,AX

        ; Copy masks A and B to display memory for each row 'r' as follows
        ; Ar0,Br1,-,-, Br0,Br1,-,-, -,-,-,-, Ar2,Br3,-,-, Br2,Br3,-,-, -,-,-,-,
        ; (where Ar2 = Mask 'A' row 'r' byte '2' (columns 16-23))
        ; Converting AND,XOR pair to A,B pair using the following formulas:
        ; A_Mask = NOT (AND_Mask XOR XOR_Mask)
        ; B_Mask = NOT AND_Mask

        MOV     CX,64                     ;for 32 rows of cursor data,
        MOV     BX,Arg_AND_Mask           ;Initialize index into source patterns
        MOV     SI,Arg_XOR_Mask
        MOV     ES,CS:Graf_Seg            ;Pointer to destination
        MOV     DI,0FE00h                 ;   use last 512 bytes of the last page
        MOV     AL,CS:Video_Pages         ;Last page
        DEC     AL
        CALL    Select_Page               ;Select last page
Load_256:
        LODSW                             ;Fetch next two bytes of XOR mask
        XOR     AX,[BX]                   ;XOR XOR mask with AND mask
        NOT     AX                        ;Negate result
        STOSW                             ;Save the result as mask A
        MOV     AX,[BX]                   ;Fetch AND Mask
        NOT     AX                        ;Negate result
        STOSW                             ;Save the result as mask B
        ADD     DI,4                      ;Skip next four bytes of destination
        ADD     BX,2                      ;Update source index
        LOOP    Load_256
        JMP     SC_Enable

        ;---------------------------------------------------------------
        ; Copy cursor masks for planar modes
        ;---------------------------------------------------------------

        ;Initialize Pattern Start Address to FE00 in last visible page
        ; Curs addr regs = 3F80 + Page SHL 14, Page:Offset=Page:FE00
SC_Do_Planar:
        MOV     AL,CURS_ADDR_MID_REG      ;Index of cursor address mid
        MOV     AH,CS:Video_Pages         ;Fetch number of visible pages
        DEC     AH                        ;Convert to page number
        ROR     AH,1                      ;Move page number to bits 4-7
        ROR     AH,1
        ADD     AH,3Fh                    ;Select last page
```

```
        OUT     DX,AX
        MOV     AL,CURS_ADDR_LOW_REG     ;Index of cursor address low
        MOV     AH,80h                   ;Select last page, last 512 bytes
        OUT     DX,AX

        ; Initialize Pattern End Address

        MOV     AL,CURS_ADDR_END_REG     ;Index of cursor address end
        ADD     AH,1Fh                   ;Set end 32 lines after start
        OUT     DX,AX

        ;Disable set/reset and enable 8 bits for write

        MOV     DX,GRAPHICS_CTRL_PORT    ;Address of graphics controller
        MOV     AX,SR_ENABLE_REG         ;Index of set/reset enable, data=0
        OUT     DX,AX                    ;Disable set/reset

        MOV     AX,BIT_MASK_REG+0FF00h   ;Index of bit mask register,data=FF
        OUT     DX,AX                    ;Enable all 8 bits for write

        ; Copy the cursor patterns (this is slow because a plane is enabled
        ; for each byte, but code is easier to understand)
        ; Plane0: Ar0,-,Ar2,-,...
        ; Plane1: Ar1,-,Ar3,-,...
        ; Plane2: Br0,-,Br2,-,...
        ; Plane3: Br1,-,Br3,-,...
        ; (where Ar2 = Mask 'A' row 'r' byte '2' (columns 16-23))
        ; Converting AND,XOR pair to A,B pair using the following formulas:
        ; A_Mask = NOT (AND_Mask XOR XOR_Mask)
        ; B_Mask = NOT AND_Mask

        MOV     CX,64                    ;for 32 rows of cursor data
        XOR     SI,SI                    ;Initialize index into source patterns
        MOV     ES,CS:Graf_Seg           ;Pointer to destination
        MOV     DI,0FE00h                ;   use last 256 bytes of the last page
        MOV     AL,CS:Video_Pages        ;Last page
        DEC     AL
        CALL    Select_Page              ;Select last page
        MOV     DX,SEQUENCER_PORT        ;Address of graphics controller
        MOV     AL,PLANE_ENABLE_REG      ;Index of plane enable register
        MOV     BX,Arg_AND_Mask          ;Initialize index into source patterns
        MOV     SI,Arg_XOR_Mask
Load_16:
        PUSH    CX                       ;Preserve counter
        MOV     AH,1                     ;Plane to enable
        OUT     DX,AX                    ;Enable plane 0 for write
        MOV     CX,[BX]                  ;Fetch next two bytes of AND mask
        XOR     CX,[SI]                  ;XOR XOR mask with AND mask
        NOT     CX                       ;Negate result
        MOV     ES:[DI],CL               ;Save next byte of mask A

        MOV     AH,2                     ;Plane to enable
        OUT     DX,AX                    ;Enable plane 1
        MOV     ES:[DI],CH               ;Save next byte of mask A

        MOV     AH,4                     ;Plane to enable
        OUT     DX,AX                    ;Enable plane 0
        MOV     CX,[BX]                  ;Fetch next two bytes of AND mask
        NOT     CX                       ;Negate the AND mask
        MOV     ES:[DI],CL               ;Save next byte of mask B

        MOV     AH,8                     ;Plane to enable
        OUT     DX,AX                    ;Enable plane 1
        MOV     ES:[DI],CH               ;Save next byte of mask B
        ADD     SI,2                     ;Update pointers
        ADD     BX,2
        ADD     DI,2
        POP     CX                       ;Restore counter
        LOOP    Load_16

        MOV     AH,0Fh                   ;Enable all planes for write
```

```
        OUT     DX,AX

        ;--------------------------------------------------------------
        ; Set cursor position to 'off-screen' and enable display of cursor
        ;--------------------------------------------------------------
SC_Enable:
        MOV     AX,CS:Video_Height          ;Set Y = Below last scan line
        PUSH    AX
        XOR     AX,AX                       ;Set X = 0
        PUSH    AX
        CALL    _Move_Cursor                ;Use proc to set cursor position
        ADD     SP,4

        MOV     DX,EXTENDED_PORT            ;Address of extened registers
        MOV     AL,GCUR_MODE                ;Index of control register
        MOV     AH,1                        ;Value for enable cursor
        OUT     DX,AX                       ;Turn cursor on

        ;--------------------------------------------------------------
        ; Clean up and return
        ;--------------------------------------------------------------

        POP     DS                          ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI
        MOV     SP,BP                       ;Restore stack
        POP     BP
        RET
_Set_Cursor     ENDP

;*********************************************************************
;*                                                                   *
;* _Move_Cursor(Curs_X, Curs_Y)                                      *
;*      This procedure is used to move the cursor from one           *
;*      location to another.                                         *
;*                                                                   *
;*********************************************************************

Arg_Curs_X      EQU     BYTE PTR [BP+4]     ;Formal parameters
Arg_Curs_Y      EQU     BYTE PTR [BP+6]

Curs_X          EQU     WORD PTR [BP-2]
Curs_Y          EQU     WORD PTR [BP-4]

_Move_Cursor    PROC    NEAR
        PUSH    BP                          ;Standard high-level entry
        MOV     BP,SP
        SUB     SP,4

        PUSH    SI                          ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Load Cursor X position registers

        MOV     DX,EXTENDED_PORT            ;Address of extened registers
        MOV     AL,GCUR_XHI_REG             ;Index of x start high register
        MOV     AH,ARG_CURS_X[1]            ;Fetch high byte
        OUT     DX,AX                       ;Write high value
        INC     AL                          ;Index of x start low register
        MOV     AH,ARG_CURS_X[0]            ;Fetch low byte
        OUT     DX,AX                       ;Write low value
        ; Load Cursor Y position registers

        MOV     DX,EXTENDED_PORT            ;Address of extened registers
        MOV     AL,GCUR_YHI_REG             ;Index of y start high reg
        MOV     AH,ARG_CURS_Y[1]            ;Fetch high byte
        OUT     DX,AX                       ;Write high value
        INC     AL                          ;Index of y start low reg
```

```
        MOV     AH,ARG_CURS_Y[0]        ;Fetch low byte
        OUT     DX,AX                   ;Write low value

        ; Clean up and return

        POP     DS                      ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI

        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
_Move_Cursor    ENDP

;**********************************************************************
;*                                                                    *
;* _Remove_Cursor                                                     *
;*      This procedure is used to remove the cursor from the screen   *
;*                                                                    *
;**********************************************************************

_Remove_Cursor PROC NEAR
        MOV     DX,EXTENDED_PORT        ;Address of extended registers
        MOV     AX,GCUR_MODE+0000h      ;AL=Index, AH=Data(turn cursor off)
        OUT     DX,AX                   ;Write new value of cursor mode
        RET
_Remove_Cursor ENDP
_TEXT   ENDS
        END
```

## Detection and Identification

Chips and Technologies recommends that 82C452 VGA chips be detected using the Global ID register (I/O address 104h) and the Version register (address 3D6h, index 00h). Global ID should always be A5h for CTI products. Code similar to that below can be used to identify CTI products:

```
; Place VGA in SETUP mode
CLI                                     ;Disable interrupts
MOV             DX,46E8h                ;Address of setup control register
IN              AL,DX                   ;Get current value
OR              AL,10h                  ;Turn setup bit on
OUT             DX,AL                   ;Place chip in setup mode
; Enable extended register bank
MOV             DX,103h                 ;Address the extended enable register
IN              AL,DX                   ;Get current value
OR              AL,80h                  ;Turn enable bit on
OUT             DX,AL                   ;Enable extended register bank
; Read Global ID
MOV             DX,104h                 ;Address of Global ID register
IN              AL,DX                   ;Read the ID
MOV             AH,AL                   ;Save ID for later
; Place VGA in NORMAL mode
MOV             DX,46E8h                ;Address the setup control register
IN              AL,DX                   ;Get current value
AND             AL,0EFh                 ;Clear setup bit
OUT             DX,AL                   ;Enable normal mode
STI                                     ;Enable interrupts
; Read version extended register
MOV             DX,3D6h                 ;Address of extended register
MOV             AL,00h                  ;Index of version register
OUT             DX,AL                   ;Select version register
INC             DX
IN              AL,DX                   ;Fetch version value
; Check if Chips 82C45x chip
```

```
            CMP     AH,5Ah          ;Look for product ID (saved earlier)
            JNE     Not_ChipsVGA    ;Quit if not Chips product
            MOV     AH,AL
            AND     AH,0F0h         ;Isolate chip id
            CMP     AL,00h          ;Check for 82C451 id
            JE      Chips_451       ;...Yes, found 451
            CMP     AL,10h          ;Check for 82C452 id
            JE      Chips_452       ;...Yes, found 452
            CMP     AL,30h          ;Check for 82C453 id
            JE      Chips_453       ;...Yes, found 453
Not_ChipsVGA:                       ;No, this is not Chips VGA
```

# 14

# Genoa 6400 Genoa SuperVGA

genoa
SYSTEMS CORPORATION

# Introduction

For Genoa VGA products, SuperVGA is more than just a nickname; it is the name under which the product is marketed. Early versions of the SuperVGA were based on the ET3000 VGA chip made by Tseng Laboratories until Genoa was able to complete the design of their own VGA chip. The product described here is the one that is based on Genoa's own VLSI VGA chip. The Tseng Labs ET3000 VGA chip, which has been used on many different VGA products, is described in Chapter 17.

The Genoa SuperVGA is sold in two standard memory configurations. The standard SuperVGA, 6300, comes equipped with 256K of display memory and will support resolutions up to 800x600 pixels with 16 colors or 640x400 pixels with 256 colors. SuperVGA models 6400, 6400A, 6600 and 6600A include 512K of display memory and will support resolutions as high as 1024x768 pixels with 16 colors or 800x600 pixels with 256 colors. The 6400 series adapters are for the AT bus and 6600 series adapters are for the IBM Micro Channel (PS/2) bus.

6400A and 6600A adapters support 70Hz vertical refresh rates (instead of 60Hz) for reduced screen flicker; this higher refresh rate is especially popular in Europe. The only difference this presents to the programmer is a faster vertical retrace interrupt. Not all displays will support this faster refresh rate.

Genoa's SuperVGA includes EGA, CGA, MDA and Hercules emulation modes, and has the ability to automatically switch to an emulation mode when software is executed that addresses a register that is specific to one of these other adapters.

To access the full display memory in high resolution modes, a memory paging mechanism allows for the selection of separate read and write pages in display memory.

# New Display Modes

Table 14-1 lists the enhanced display modes that are supported by the Genoa SuperVGA. All modes can be selected via BIOS function 0 (Mode Select). Genoa SuperVGA boards include configuration switches used to indicate the type of display attached. The BIOS uses the switch settings to determine if a given mode is possible on the indicated display, and will abort the selection if not. Care must be taken if the switches are set for IBM 80xx and 81xx displays since for those displays the BIOS will use the monitor id lines from the display to automatically determine the display model. Not all VGA-compatible displays include monitor ID lines.

Table 14-1. Enhanced display modes—Genoa SuperVGA

| Mode | Type | Resolution | Colors | Memory Required | Display Type |
|---|---|---|---|---|---|
| 43h | Text | 80 col x 20 rows | mono | 256 KB | VGA |
| 44h | Text | 80 col x 32 rows | mono | 256 KB | VGA |
| 45h | Text | 80 col x 44 rows | mono | 256 KB | VGA |
| 46h | Text | 132 col x 25 rows | mono | 256 KB | VGA |
| 47h | Text | 132 col x 29 rows | mono | 256 KB | VGA |
| 48h | Text | 132 col x 32 rows | mono | 256 KB | VGA |
| 49h | Text | 132 col x 44 rows | mono | 256 KB | VGA |
| 58h | Text | 80 col x 32 rows | 16 | 256 KB | VGA |
| 60h | Text | 132 col x 25 rows | 16 | 256 KB | VGA |
| 61h | Text | 132 col x 29 rows | 16 | 256 KB | VGA |
| 62h | Text | 132 col x 32 rows | 16 | 256 KB | VGA |
| 63h | Text | 132 col x 44 rows | 16 | 256 KB | VGA |
| 64h | Text | 132 col x 60 rows | 16 | 256 KB | VGA |
| 72h | Text | 80 col x 60 rows | 16 | 256 KB | VGA |
| 74h | Text | 80 col x 66 rows | 16 | 256 KB | VGA |
| 78h | Text | 100 col x 75 rows | 16 | 256 KB | VGA |
| 5Ch | Graphics | 640x480 | 256 | 512 KB | VGA |
| 5Eh | Graphics | 800x600 | 256 | 512 KB | Super VGA |
| 5Fh | Graphics | 1024x768 | 16 | 512 KB | 8514 or XL |
| 73h | Graphics | 640x480 | 16 | 256 KB | VGA |
| 79h | Graphics | 800x600 | 16 | 256 KB | Super VGA |
| 7Dh | Graphics | 512x512 | 256 | 256 KB | Super VGA |
| 7Eh | Graphics | 640x400 | 256 | 256 KB | VGA |
| 7Fh | Graphics | 1024x768 | 4 | 256 KB | 8514 or XL |

# Memory Organization

For all extended display modes of the SuperVGA, display memory organization is closely patterned after standard IBM VGA display modes. Genoa SuperVGA includes a display memory paging mechanism that is needed in some display modes to make the entire display memory accessible to the processor. Display memory paging is described in detail later in this chapter.

## High Resolution Text Modes

These modes utilize memory maps that are similar to those used in standard text modes (modes 0,1,2,3 and 7), except that the number of characters per line and/or

number of lines per screen is increased. Display memory is organized as shown in Figure 5-1 (see Chapter 5).

## 256-Color Graphics Modes

Memory organization for these modes resembles VGA mode 13h (320x200 256-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is shown in Figure 8-1. See Chapter 8 for programming examples.

## 16-Color Graphics Modes

Memory organization for these modes resembles VGA mode 12h (640x480 16-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is shown in Figure 7-1. See Chapter 7 for programming examples.

## 4-Color Graphics Modes

Memory organization for mode 7F, the 1024x768 four-color mode, is similar to that of VGA mode 12h (640x480 16-color graphics) except that only planes 0 and 1 are used. See the section "Two Consecutive Planes" in Chapter 9 for programming examples.

# New Registers

Genoa has added extended registers to the CRT Controller, Sequencer and Graphics Controller to implement the extended functions of the SuperVGA. Table 14-2 lists these new registers.

**Table 14-2. SuperVGA extended registers**

| Address | Index | Description |
|---|---|---|
| 3B4h/3D4h | 2Fh | Interlace Control Register |
| 3B4h/3D4h | 2Eh | Herchi Register |
| 3C4h | 05h | Configuration Register |
| 3C4h | 06h | Memory Page Select Register |
| 3C4h | 07h | Enhanced Control 2 |
| 3C4h | 08h | Enhanced Control 3 |
| 3C4h | 10h | Enhanced Control 4 |
| 3CEh | 0Ah | Program Status Register 1 |
| 3CEh | 0Bh | Program Status Register 2 |

## Interlace Control Register (I/O Address 3B4h/3D4h Index 2Fh)

D3 to D7 - Reserved
D2 - Select character clock as memory addressing counter clock
D1 - Enable quadword addressing mode
D0 - Enable Interlacing (1 = interlace)

## Herchi Register (I/O Address 3B4/3D4 Index 2Eh)

D2 to D7 - Reserved
D1 - Enable Chinese application under Hercules mode
D0 - Enable maximum scan line register under CGA mode

## Configuration Register (I/O Address 3C4h Index 5)

D7 - Enable 8 simultaneous fonts (1 = enabled)
D5,D6 - BIOS size (00 = 24k, 01 = 30k, 10 = 32k. 11 = 0k)
D4 - Enable 3XX addressing (instead of 2XX). Read-only bit
D3 - Reserved
D2 - Enable 8-bit BIOS (instead of 16-bit). Read-only bit
D1 - Enable 8-bit bus (instead of 16-bit). Read-only bit
D0 - Enable XT/AT operation (instead of Micro Channel). Read-only bit

## Memory Page Select Register (I/O Address 3C4h Index 6)

D7 - Reserved
D6 - Memory Paging Configuration
D5-D3 - Write Page Select
D2-D0 - Read Page Select

Write Page Select and Read Page Select select display memory pages for reading and writing. Pages are either 64K or 128K in length, depending on the host window size defined in the miscellaneous Register of the Graphics Controller.

Memory Paging Configuration, when set to zero, causes the least significant bit of the read and write page select fields to be replaced by the Odd/Even Page Select bit (D3 of the Miscellaneous Output register - I/O address 3C2h).

## Enhanced Control Register 2 (I/O Address 3C4h Index 7)

D7 - Reserved
D6 - NMI enable
D5 - Enable TTL monitor output
D4 - Reserved
D3 - Motherboard implementation (instead of add-on)
D2 - Enable 16-bit memory R/W
D1 - Allow frequencies above 50MHz
D0 - External clock select (bit D2 of 3-bit value)

## Enhanced Control Register 3 (I/O Address 3C4h Index 8)

D7 - Enable 1024x768 addressing
D6 - Enable extended memory addressing
D5 - Enable chain 8 addressing
D4 - Disable flicker-free function
D3 - Enable EGA function
D2 - Enable autoswitch through 3D8
D1 - Enable autoswitch through 3B8
D0 - Set emulation modes (MDA, CGA, Hercules)

## Enhanced Control Register 4 (I/O Address 3C4h Index 10h)

D7 - Select memory bank 1
D6 - Enable fast write
D5 - Reserved
D4 - Reserved
D3 - Enable pre_wait function
D2 - Enable two bank memory access
D1 - Enable fast access function
D0 - Enable fast scroll function

## Program Status Registers 1 and 2 (I/O Address 3CEh Index 0Ah and 0Bh)

These are general purpose 8-bit read/writable registers for temporary data storage.

# Programming Examples

## Display Memory Paging

The display memory paging mechanism of the SuperVGA maps selected portions of the display memory to the processor. Operation of display memory paging is very similar to the paging mechanism used for expanded memory boards (also called EMS or LIM memory). A 64K or 128K logical page of VGA RAM (a chunk of display memory) is mapped into the PC host address space in the normal VGA display memory address space. An I/O register (the Memory Page Select register), located in the Sequencer at index 6, is used to define which pages of display memory are selected. This is illustrated in Figure 14-1. For programming examples showing how to select pages see the routines Select_Page, Select_Read_Page, and Select_Write_Page in Listing 14-1.

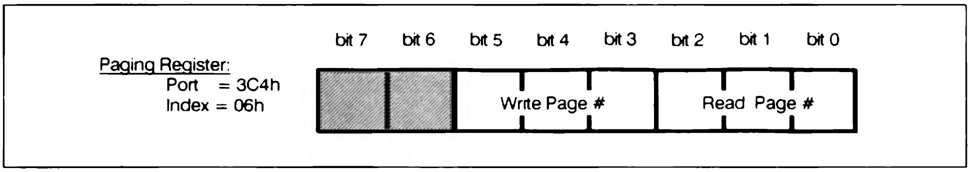


**Figure 14-1.** **Memory paging**

**Listing 14-1.**

```
File:  GENOA\SELECT.ASM
;*****************************************************************************
;* File:   SELECT.ASM                                                         *
;* Description: This module contains procedures to select mode and to        *
;*              select pages.  It also initializes global variables          *
;*              according to the values in the MODE.INC include file.        *
;* Entry Points:                                                              *
;*              _Select_Graphics     - Select a graphics mode                 *
;*              _Select_Text         - Set VGA adapter into text mode         *
;*              _Select_Page         - Set read and write page                *
;*              _Select_Read_Page    - Select read page only                  *
;*              _Select_Write_Page   - Select write page only                 *
;* Uses:                                                                      *
;*              MODE.INC      - Mode dependent constants                      *
;*              Following are modes and paths for Genoa 6400 boards:          *
;*      |----- 256 colors ------|  |-- 16 colors --|  4 colors  2 colors *
;*      640x400  640x480  800x600  800x600  1024x768  1024x768  1024x768 *
;* Mode:  7Eh      5Ch      5Eh     6Ah(79h)    5Fh       7Fh       N/A     *
;* Path: 256COL  256COL   256COL   16COL      16COL     4COL01     N/A     *
;*****************************************************************************

       INCLUDE VGA.INC
       INCLUDE MODE.INC              ;Mode dependent constants

       PUBLIC    _Select_Graphics
       PUBLIC    _Select_Text
       PUBLIC    _Select_Page
       PUBLIC    _Select_Read_Page
       PUBLIC    _Select_Write_Page
```

```
        PUBLIC    Select_Page
        PUBLIC    Select_Read_Page
        PUBLIC    Select_Write_Page
        PUBLIC    Enable_Dual_Page
        PUBLIC    Disable_Dual_Page

        PUBLIC    Graf_Seg
        PUBLIC    Video_Height
        PUBLIC    Video_Width
        PUBLIC    Video_Pitch
        PUBLIC    Video_Pages
        PUBLIC    Ras_Buffer
        PUBLIC    Two_Pages

        PUBLIC    Last_Byte

;-----------------------------------------------------------------------
; Data segment variables
;-----------------------------------------------------------------------

;_DATA   SEGMENT WORD PUBLIC 'DATA'
;_DATA   ENDS

;-----------------------------------------------------------------------
; Constant definitions
;-----------------------------------------------------------------------

PAGE_SEL_PORT    EQU   3C4h           ;IO Address for page select register
PAGE_SEL_INDEX   EQU   6              ;Index for page select register

;-----------------------------------------------------------------------
; Code segment variables
;-----------------------------------------------------------------------

_TEXT     SEGMENT BYTE PUBLIC 'CODE'

Graf_Seg         DW    0A000h         ;Graphics segment addresses
                 DW    0A000h
OffScreen_Seg    DW    0A000h         ;First byte beyond visible screen
Video_Pitch      DW    SCREEN_PITCH   ;Number of bytes in one raster
Video_Height     DW    SCREEN_HEIGHT  ;Number of rasters
Video_Width      DW    SCREEN_WIDTH   ;Number of pixels in a raster
Video_Pages      DW    SCREEN_PAGES   ;Number of pages in the screen
Ras_Buffer       DB    1024 DUP (0)   ;Working buffer
R_Page           DB    0FFh           ;Most recently selected page
W_Page           DB    0FFh
RW_Page          DB    0FFh
Two_Pages        DB    CAN_DO_RW      ;Indicate separate R & W capability

;***********************************************************************
;*                                                                     *
;* _Select_Graphics(HorizPtr, VertPtr, ColorsPtr)                      *
;*      Initialize VGA adapter to 640x400 mode with                    *
;*      256 colors.                                                    *
;*                                                                     *
;* Entry:                                                              *
;*      None                                                           *
;*                                                                     *
;* Returns:                                                            *
;*      VertPtr   - Vertical resolution                                *
;*      HorizPtr  - Horizontal resolution                              *
;*      ColorsPtr - Number of supported colors                         *
;*                                                                     *
;***********************************************************************

Arg_HorizPtr   EQU   WORD PTR [BP+4]  ;Formal parameters
Arg_VertPtr    EQU   WORD PTR [BP+6]  ;Formal parameters
Arg_ColorsPtr  EQU   WORD PTR [BP+8]  ;Formal parameters

_Select_Graphics PROC NEAR
        PUSH   BP                     ;Standard C entry point
```

```
        MOV     BP,SP

        PUSH    DI                      ;Preserve segment registers
        PUSH    SI
        PUSH    DS
        PUSH    ES

        ; Select graphics mode

        MOV     AX,GRAPHICS_MODE        ;Select graphics mode
        INT     10h

        ; Reset 'last selected page'

        MOV     AL,0FFh                 ;Use 'non-existent' page number
        MOV     CS:R_Page,AL            ;Set currently selected page
        MOV     CS:W_Page,AL
        MOV     CS:RW_Page,AL

        ; Set return parameters

        MOV     SI,Arg_VertPtr          ;Fetch pointer to vertical resolution
        MOV     WORD PTR [SI],SCREEN_HEIGHT   ;Set vertical resolution
        MOV     SI,Arg_HorizPtr         ;Fetch pointer to horizontal resolution
        MOV     WORD PTR [SI],SCREEN_WIDTH    ;Set horizontal resolution
        MOV     SI,Arg_ColorsPtr        ;Fetch pointer to number of colors
        MOV     WORD PTR [SI],SCREEN_COLORS   ;Set number of colors

        ; Clean up and return to caller

        POP     ES                      ;Restore segment registers
        POP     DS
        POP     SI
        POP     DI

        MOV     SP,BP                   ;Standard C exit point
        POP     BP
        RET
_Select_Graphics ENDP

;*************************************************************************
;                                                                        *
; Select_Page                                                            *
; Entry:                                                                 *
;       AL - Page number                                                 *
;                                                                        *
;*************************************************************************

Select_Page     PROC NEAR
        CMP     AL,CS:RW_Page           ;Check if already selected
        JNE     SP_Go
        RET
SP_Go:
        PUSH    AX
        PUSH    DX

        AND     AL,7                    ;Map into range
        MOV     CS:RW_Page,AL           ;Save as latest selection
        MOV     CS:R_Page,0FFh          ;Invalidate read and write pages
        MOV     CS:W_Page,0FFh
        MOV     AH,AL                   ;Copy into bits 0-2 and 3-5
        SHL     AL,1
        SHL     AL,1
        SHL     AL,1
        OR      AH,AL

        MOV     DX,PAGE_SEL_PORT        ;Select the register
        MOV     AL,PAGE_SEL_INDEX
        OUT     DX,AL
        INC     DX
        IN      AL,DX                   ;Read in current data value
```

```
        AND     AL,80h
        OR      AL,40h
        OR      AL,AH                   ;Combine with page numbers
        OUT     DX,AL                   ;Select new page number

        POP     DX
        POP     AX
        RET
Select_Page     ENDP

;**********************************************************************
;                                                                     *
; Select_Read_Page                                                    *
; Entry:                                                              *
;       AL - Page number                                              *
;                                                                     *
;**********************************************************************

Select_Read_Page PROC NEAR
        CMP     AL,CS:R_Page            ;Check if already selected
        JNE     SRP_Go
        RET

SRP_Go:
        PUSH    AX
        PUSH    DX
        AND     AL,7                    ;Force page number into 0-7
        MOV     AH,AL                   ;Copy page number into AH
        MOV     CS:R_Page,AH            ;Save most recently selected page
        MOV     CS:RW_Page,0FFh

        MOV     DX,PAGE_SEL_PORT        ;Select the page select register
        MOV     AL,PAGE_SEL_INDEX
        OUT     DX,AL
        INC     DX
        IN      AL,DX                   ;Get current value of page select reg
        JMP     $+2
        JMP     $+2
        AND     AL,38h                  ;Preserve bits 3-5
        OR      AL,40h                  ;Force bits 6 and 7
        OR      AL,AH                   ;Move page number into 'write' bits
        OUT     DX,AL                   ;Write out the new page select
        ; Clean up and return
        POP     DX
        POP     AX
        RET
Select_Read_Page ENDP

;**********************************************************************
;                                                                     *
; Select_Write_Page                                                   *
; Entry:                                                              *
;       AL - Page number                                              *
;                                                                     *
;**********************************************************************

Select_Write_Page PROC NEAR
        CMP     AL,CS:W_Page            ;Check if already selected
        JNE     SWP_Go
        RET
SWP_Go:
        PUSH    AX                      ;Preserve page number (AX gets trashed)
        PUSH    DX
        AND     AL,7                    ;Force page number into 0-7
        MOV     AH,AL                   ;Copy page number into AH
        SHL     AH,1                    ;Move page number into bits 3-5
        SHL     AH,1
        SHL     AH,1
        MOV     CS:W_Page,AL            ;Save most recently selected page
        MOV     CS:RW_Page,0FFh
        MOV     DX,PAGE_SEL_PORT        ;Select page select register
```

```
        MOV     AL,PAGE_SEL_INDEX
        OUT     DX,AL
        INC     DX
        IN      AL,DX                   ;Get current value of page sel register
        JMP     $+2
        JMP     $+2
        AND     AL,07h                  ;Preserve bits 0-2
        OR      AL,40h                  ;Force bits 6 & 7
        OR      AL,AH                   ;Move page number into 'write' bits
        OUT     DX,AL                   ;Write out the new page select
        ; Clean up and return
        POP     DX
        POP     AX
        RET
Select_Write_Page ENDP

;**************************************************************************
;*                                                                        *
;* Enable_Dual_Page                                                       *
;* Disable_Dual_Page                                                      *
;*      Not supported by Genoa based boards                               *
;*                                                                        *
;**************************************************************************

Enable_Dual_Page    PROC NEAR
        RET
Enable_Dual_Page    ENDP

Disable_Dual_Page   PROC NEAR
        RET
Disable_Dual_Page   ENDP

;**************************************************************************
;                                                                         *
; _Select_Page(PageNumber)                                                *
; _Select_Read_Page(PageNumber)                                           *
; _Select_Write_Page(PageNumber)                                          *
; Entry:                                                                  *
;       PageNumber - Page number                                          *
;                                                                         *
;**************************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Page    PROC NEAR
        PUSH    BP                      ;Setup frame pointer
        MOV     SP,BP
        MOV     AL,Arg_PageNumber       ;Fetch argument
        POP     BP                      ;Restore BP
        JMP     Select_Page
_Select_Page    ENDP

_Select_Read_Page   PROC NEAR
        PUSH    BP                      ;Setup frame pointer
        MOV     SP,BP
        MOV     AL,Arg_PageNumber       ;Fetch argument
        POP     BP                      ;Restore BP
        JMP     Select_Read_Page
_Select_Read_Page   ENDP

_Select_Write_Page  PROC NEAR
        PUSH    BP                      ;Setup frame pointer
        MOV     SP,BP
        MOV     AL,Arg_PageNumber       ;Fetch argument
        POP     BP                      ;Restore BP
        JMP     Select_Write_Page
_Select_Write_Page  ENDP
```

```
;************************************************************************
;*                                                                      *
;* _Select_Text                                                         *
;*      Set VGA adapter to text mode                                    *
;*                                                                      *
;************************************************************************

_Select_Text    PROC NEAR
        MOV     AX,TEXT_MODE            ;Select mode 3
        INT     10h                     ;Use BIOS to reset mode
        RET
_Select_Text    ENDP

Last_Byte:
_Text   ENDS
        END
```

# Detection and Identification

Genoa recommends that their boards be detected using signature bytes in the BIOS ROM. ROM address C000:0037 contains a double word pointer that points to the location of the signature bytes (normally C000:00B4h). The signature bytes are illustrated in Table 14-3.

**Table 14-3. Genoa ID bytes**

| Address | Size | Content |
|---|---|---|
| C000:0037h | DWORD | Address of ID bytes (normally C000:00B4) |
| Id Address | 4 BYTES | 77h, 11h, 99h, 66h |

# 15

# Headland HT-208 (V7VGA) Headland Video Seven VGA1024i

Headland
VIDEO SEVEN
G2

# Introduction

In the past, Video Seven has purchased both EGA and VGA chips from several different sources for use on their products, though these chips were sometimes designed and built to Video Seven's specifications. Video Seven purchased video chips from Chips and Technologies and later from Cirrus Logic before merging with G2 to form Headland Technology. Headland Technology now manufactures the HT-208 chip, initially introduced as V7VGA, for their VGA boards. The name Video Seven has been retained only as a product name for Headland Technology's video products.

This chapter describes the Video Seven VGA1024i, a VGA adapter that is based on the HT-208 (V7VGA) chip; unless noted otherwise, all information also applies to the Video Seven FastWrite VGA (now discontinued), and the Video Seven VRAM VGA.

In addition to VGA compatibility, Video Seven products include EGA, CGA, MDA and Hercules emulation modes, high resolution graphics display modes, and a hardware graphics cursor for planar modes. Video output is analog only (TTL displays are not supported).

Headland also supplies application software drivers for programs such as MS-Windows, GEM and Ventura Publisher.

# New Display Modes

Table 15-1 on page 359 lists the enhanced display modes that are supported by the HT-208 (V7VGA). Any of the standard modes can be selected by issuing a BIOS mode select command. Enhanced modes are selected with a new BIOS service 6Fh (see section on BIOS later in this chapter).

Table 15-1. Enhanced display modes—Video Seven boards

| Mode | Type | Resolution | Colors | Memory Required | Display Type |
|---|---|---|---|---|---|
| 40h | Text | 80 col x 43 rows | 16 | 256K | VGA |
| 41h | Text | 132 col x 25 rows | 16 | 256K | VGA |
| 42h | Text | 132 col x 43 rows | 16 | 256K | VGA |
| 43h | Text | 80 col x 60 rows | 16 | 256K | VGA |
| 44h | Text | 100 col x 60 rows | 16 | 256K | VGA |
| 45h | Text | 132 col x 28 rows | 16 | 256K | VGA |
| 60h | Graphics | 752x410 | 16 | 256K | VGA |
| 61h | Graphics | 720x540 | 16 | 256K | Super VGA |
| 62h | Graphics | 800x600 | 16 | 256K | Super VGA |
| 63h | Graphics | 1024x768 | 2 | 256K | 8514 |
| 64h | Graphics | 1024x768 | 4 | 256K | 8514 |
| 65h | Graphics | 1024x768 | 16 | 512K | 8514 |
| 66h | Graphics | 640x400 | 256 | 256K | VGA |
| 67h | Graphics | 640x480 | 256 | 512K | VGA |
| 68h (1) | Graphics | 720x540 | 256 | 512K | Super VGA |
| 69h (1) | Graphics | 800x600 | 256 | 512K | XL |

NOTE. Modes 68h and 69h are supported on V-RAM VGA only, they are not supported on 1024i VGA.

# Memory Organization

For all extended display modes of the VGA1024i, display memory organization is closely patterned after standard IBM VGA display modes.

VGA1024i includes a display memory paging mechanism that is needed in some display modes to make the entire display memory accessible to the processor. Display memory paging is described in detail later in this chapter.

## High Resolution Text Modes

These modes utilize memory maps that are similar to those used in standard text modes (modes 0,1,2,3 and 7), except that the number of characters per line, or number of lines per screen, is increased. Display memory is organized as shown in Figure 5-1 (see Chapter 5).

## 2-Color Graphics Mode

Memory organization for mode 63h resembles VGA mode 11h, except that both number of pixels per scan line and number of scan lines is increased. Two 64K pages are needed in this mode.

## 4-Color Graphics Mode

Memory organization for mode 64h resembles VGA mode 12h, except that both number of pixels per scan line and number of scan lines is increased, and only two planes are used for each pixel. Two 64K pages are needed in this mode. Display memory organization is shown in Figure 9-1. See the section "Four Planes" in Chapter 9 for programming examples.

## 16-Color Graphics Modes

Memory organization for these modes resembles VGA mode 12h (640x480 16-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is shown in Figure 7-1. See Chapter 7 for programming examples.

## 256-Color Graphics Modes

Memory organization for these modes resembles VGA mode 13h (320x200 256-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is shown in Figure 8-1. See Chapter 8 for programming examples.

# New Registers

A bank of extended registers internal to the HT-208 (V7VGA) is used to access the advanced features of the adapter. The extended register bank is located mapped at the same I/O address as the Sequencer (using register indexes 6 through 7, and indexes 80h through FFh). Most of the registers in the extended bank have read and write capability. Table 15-2 shows the extended register set of the HT-208 (V7VGA). When accessing extended register bank, Headland Technology recommends that the following rules be observed:

- Before first access to extended registers, enable the access by writing value EAh to index 6 in the sequencer (3C4h).

- Disable access to extended registers whenever possible, e.g. when access is not needed, by writing value AEh to index 6 in the sequencer (3C4).
- Always restore extended registers when done, or at least set them to a 'non-disruptive' value (generally zero). BIOS mode select does not always reset extended registers.
- Avoid modifying extended registers other than the following: 94h, 9C through A3h, A5h, F1h through F6, F9h through FC, and FEh.

**Table 15-2. Extended registers—VGA1024i**

| I/O Address | Index | Register |
|---|---|---|
| 3C5 | 6 | Extension Control Register |
| 3C5 | 7 | Reset Horizontal Character Counter |
| 3C5 | 80h-82h | Test |
| 3C5 | 83h | Attribute Controller Index |
| 3C5 | 84h-8Dh | Reserved |
| 3C5 | 8Eh & 8Fh | VGA Chip Revision Level |
| 3C5 | 90h-93h | Reserved |
| 3C5 | 94h | Pointer Pattern Address |
| 3C5 | 95h-9Bh | Reserved |
| 3C5 | 9Ch | Pointer Horizontal Position High |
| 3C5 | 9Dh | Pointer Horizontal Position Low |
| 3C5 | 9Eh | Pointer Vertical Position High |
| 3C5 | 9Fh | Pointer Vertical Position Low |
| 3C5 | A0h | GC Memory Latch 0 |
| 3C5 | A1h | GC Memory Latch 1 |
| 3C5 | A2h | GC Memory Latch 2 |
| 3C5 | A3h | GC Memory Latch 3 |
| 3C5 | A4h | Clock Select |
| 3C5 | A5h | Cursor Attributes |
| 3C5 | A6h-AFh | Reserved |
| 3C5 | B0h-BFh | Scratch Registers |
| 3C5 | C0h-E9h | Reserved |
| 3C5 | EAh | Switch Strobe |
| 3C5 | EBh | Emulation Control |
| 3C5 | ECh | Foreground Latch 0 |
| 3C5 | EDh | Foreground Latch 1 |
| 3C5 | EEh | Foreground Latch 2 |
| 3C5 | EFh | Foreground Latch 3 |
| 3C5 | F0 | Fast Foreground Latch Load |
| 3C5 | F1 | Fast Latch Load State |
| 3C5 | F2 | Fast Background Latch Load |
| 3C5 | F3 | Masked Write Control |
| 3C5 | F4 | Masked Write Mask |

Table 15-2. Extended registers—VGA1024i *(continued)*

| I/O Address | Index | Register |
|---|---|---|
| 3C5 | F5 | Foreground/Background Pattern |
| 3C5 | F6 | 1 MB RAM Bank Select |
| 3C5 | F7 | Switch Readback |
| 3C5 | F8 | Extended Clock Control |
| 3C5 | F9 | Extended Page Select |
| 3C5 | FA | Extended Foreground Color |
| 3C5 | FB | Extended Background Color |
| 3C5 | FC | Compatibility Control |
| 3C5 | FD | Extended Timing Select |
| 3C5 | FE | Foreground/Background Control |
| 3C5 | FF | 16-bit Interface Control |

## Index 6 - Extension Control Register

D7-D1 - unused
D0 - Extensions Access Enable

Extensions Access Enable must be set before the extended register bank (indexes 80h - FFh) can be written to or read from. This bit is normally set in extended modes by the BIOS mode select function.

## Index 1Fh - Identification Register

This read-only register will read back the current value of the CRT Controller Start Address High register (I/O address 3B5/3D5, index C), exclusive-ored with the constant value EAh. This register can be used to detect the presence of the V7VGA chip (see the programming examples for more details).

## Index 8Eh and Index 8Fh - VGA Chip Revision Register

This is an 8-bit register which is redundantly mapped at index 8Eh and 8Fh. Headland Technology has defined the following values for this register:

| | |
|---|---|
| 70h | V7VGA chip revisions 1,2, or 3 |
| 71h | V7VGA chip revision 4 |
| 72h-7Fh | Reserved for future versions of V7VGA |
| 80h-FFh | VEGA VGA chip |
| 0-6Fh | Reserved for future Video Seven products |

## Hardware Graphics Cursor

Graphics cursors are used extensively in graphical interfaces where a pointing device such as a mouse or trackball is used to position an icon (usually an arrow) on the screen. For most VGA adapters, graphics cursors must be implemented in software. Hardware cursor support can reduce the burden on the processor and improve performance.

VGA1024i provides hardware cursor support for all planar graphics modes (modes 0Fh through 12h, and modes 60h through 65h). The graphics cursor is a 32 pixel x 32 pixel programmable pattern that is superimposed on the screen. It is defined, controlled and positioned using registers 94h, 9Ch through 9Fh, and FFh, in the extended register bank.

Two 128-byte blocks form a 256-byte pattern in display memory which defines the shape of the graphics cursor. Cursor pattern data, which consists of a 128-byte AND mask followed by a 128-byte XOR mask, is stored in off-screen display memory anywhere from memory address offset C000h through FFC0h, and may reside in any one of up to four possible banks of display memory (in planar modes). The Pointer Pattern Address register (index 94h) is used to define the starting address for cursor pattern data as shown in Figure 15-3. In addition, the 16-bit Interface Control register (index FFh) can be used to select which 64K page of display memory graphics cursor data will be read from. Registers 9Ch through 9Fh can be used to define the position of the cursor on the screen.

### *Index 94h - Pointer Pattern Address Register*

D7 - A13
D6 - A12
D5 - A11
D4 - A10
D3 - A9
D2 - A8
D1 - A7
D0 - A6

This register together with bits D5 and D6 of the 16-bit Interface Control register determine the address of the cursor pattern in display memory (see Figure 15-3 on page 380).

### *Index 9Ch - Pointer Horizontal Position High*

D7-D3 - Unused
D2 - Horizontal position bit 10

D1 - Horizontal position bit 9
D0 - Horizontal position bit 8

### Index 9Dh - Pointer Horizontal Position Low

D7 - Horizontal position bit 7
D6 - Horizontal position bit 6
D5 - Horizontal position bit 5
D4 - Horizontal position bit 4
D3 - Horizontal position bit 3
D2 - Horizontal position bit 2
D1 - Horizontal position bit 1
D0 - Horizontal position bit 0

This register together with Pointer Horizontal Position High determine the X coordinate of the cursor.

### Index 9Eh - Pointer Vertical Position High

D7-D2 - Unused
D1 - Vertical position bit 9
D0 - Vertical position bit 8

### Index 9Fh - Pointer Vertical Position Low

D7 - Vertical position bit 7
D6 - Vertical position bit 6
D5 - Vertical position bit 5
D4 - Vertical position bit 4
D3 - Vertical position bit 3
D2 - Vertical position bit 2
D1 - Vertical position bit 1
D0 - Vertical position bit 0

This register together with Pointer Horizontal Position High determine the X coordinate of the cursor.

### Index A5 - Cursor Attributes Register

This register controls attributes of both the standard text cursor and the 32x32 graphics cursor.

D7 Graphics Cursor Enable (1 = enabled)
D6-D4 Unused

D3 Text Cursor Mode (0 = replace, 1 = XOR)
D2,D1 unused
D0 Cursor blink disable (1 = disabled)

### *Index FFh - 16-bit Interface Control Enable*

D7 Reserved
D5-D6 Cursor pattern page select
D0-D4 Reserved

This register together with Pointer Pattern Address register determine the location of cursor pattern in the display memory.

## Index A0h through A3h - Graphics Controller Data Latches

All EGA and VGA products include data latches internal to the Graphics Controller that can be used to perform logical functions on display data. These latches are not directly accessible by the processor on either EGA or VGA. VGA1024i, however, has made these latches available for both reading and writing through the extended register bank at these indexes:

Plane 0 Memory Latch - I/O Address 3C5 Index A0
Plane 1 Memory Latch - I/O Address 3C5 Index A1
Plane 2 Memory Latch - I/O Address 3C5 Index A2
Plane 3 Memory Latch - I/O Address 3C5 Index A3

These registers can be used, when different data bytes need to be loaded into each of the four planes, to avoid excessive plane enable/disable. The same registers can also be accessed by four successive writes to extended register F0h. The SetCursor routine shown in listing 15-2 uses this capability.

## Foreground/Background Operations

VGA1024i has two new Graphics Controller modes to speed drawing operations. One of the new modes performs color expansion of a monochrome bitmap in hardware and the other provides hardware support for dithering. The two operations are illustrated in Figure 15-1 on page 366.

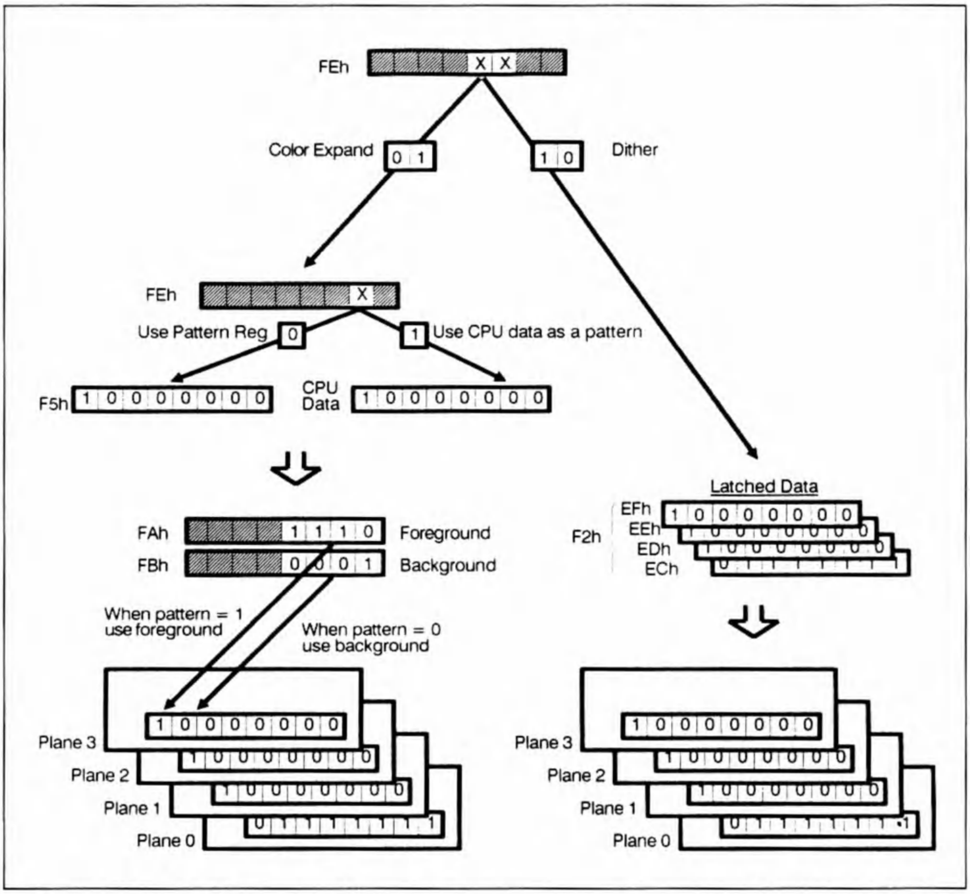


Figure 15-1. Color Expand and Dither registers

## Index ECh through EFh - Foreground Latch

These four registers are used to define the foreground latch.

## Index F0h - Fast Foreground Latch

Write operations to the Fast Foreground Pattern register (Index F0) can be used to sequentially load all four foreground pattern registers (ECh through EFh). Four processor writes to this address will cause all four registers to be loaded. Read from F0h will reset to first register.

## Index F1h - Fast Latch Load State Register

D7,D6 - Unused
D5,D4 - Foreground Latch Load State
D3,D2 - Unused
D1,D0 - Background Latch Load State

Foreground Latch Load State defines which foreground latch (index ECh-EFh) will be written by the next write to index F0h. This value is automatically reset to zero by a read operation at index F0h.

Background Latch Load State defines which background latch will be written by the next write to index F2h. This value is automatically reset to zero by a read operation at index F2h.

## Index F2h - Fast Background Pattern

Write operations to the Fast Background Pattern Register (Index F2) can be used to sequentially load all four of the normal VGA Processor Read Latches. Four processor writes to this address will cause all four registers to be loaded. A read from this address will reset to the first register.

## Index FAh - Foreground Color and Index FBh - Background Color

These registers determine foreground and background colors in the color expansion mode.

## Index FEh - Foreground/Background Control Register

| | |
|---|---|
| D7-D4 | unused |
| D2-D3 | foreground/background mode select:<br>00 - Standard VGA Mode<br>01 - Color Expansion Mode<br>10 - Dithered Foreground Mode<br>11 - invalid |
| D1 | foreground/background source select |
| D0 | unused |

**Color Expansion Mode.** In color expansion mode, data is written to all four color planes simultaneously. For each bit of write data from the processor, a zero bit causes the four-bit background color stored in the Background Color register (Index FBh) to be written into the four color planes at that pixel position. A one bit causes the four-bit foreground color stored in the Foreground Color register (Index FAh) to be written

into the four color planes at that pixel position. This permits eight pixels of a monochrome display pattern to be color expanded into two colors in a single memory cycle.

**Dithered Foreground Mode.** In dithered foreground mode, four Foreground Pattern registers (Index EC-EF), one for each plane, are used in place of processor write data to the Graphics Controller. These registers can be loaded with a dithering pattern. The normal VGA Processor Read Latches function as usual, and may be used to logically combine screen data with the dithering pattern.

**Foreground/Background Source Select.** This selects the source of the pattern for color expansion to either be processor write data if D1 equals 1 or data from the Foreground/Background Pattern register (Index F5h) if D1 equals 0.

## Display Memory Paging

V7VGA uses a 4-bit page number for memory page selection. These bits are in three separate registers, and page number read back is done by a different method than is used to select page numbers (see Figure 15-2 on page 373). Headland Technology recommends that paging first be enabled by setting bit D2 in the Paging Control register (index FCh), even though this is normally done by the BIOS mode select function.

### *Index FCh - 256-Color Paging Control Register*

D7 - Enable 3C3
D6 - Reserved
D5 - Reserved
D4 - Reserved
D3 - Reserved
D2 - 256-Color Paging Enable
D1 - 256-Color 64KB/128KB Paging Select
D0 - Reserved

When Enable 3C3 is set to one, I/O port 3C3h can be used to enable and disable all I/O and memory operations to the VGA. When Enable 3C3 is zero, I/O port 3C3h has no effect on the VGA.

If 256-Color Paging Enable is set to one, then one of two display memory paging modes will be in effect, as explained below.

If 256-Color 64KB/128KB Paging Select is set to zero, and 256-Color Paging Enable is set to one, then 64K paging is selected. Display memory can be accessed as four 64K pages, with page selection performed by bit D5 of the Miscellaneous Output register (I/O Address 3C2) and bit D0 of the 256-Color Paging register (see below).

If 256-Color 64 KB/128 KB Paging Select is set to one, and 256-Color Paging Enable is set to one, then 128K paging is selected. Display memory can be accessed as two 128K

pages, with page selection performed by bit D5 of the Miscellaneous Output register (I/O Address 3C2).

In extended graphics modes this register is normally initialized by the BIOS to 64K pages, with paging enabled.

### Index F9h - 256-Color Paging Register

D7-D1 - Unused
D0 - 256 Color Extended Page Select

If 256-Color 64K paging is selected (see register FCh above), then this bit is used as one of three page select bits for display memory.

### Index F6h - Bank Select Register

D7 - Line Compare Bank Reset
D6 - Counter Bank Enable
D5 - CRTC Read Bank Select 1
D4 - CRTC Read Bank Select 0
D3 - CPU Read Bank Select 1
D2 - CPU Read Bank Select 0
D1 - CPU Write Bank Select 1
D0 - CPU Write Bank Select 0

Of interest for page selection are bits D2-D3 and bits D0-D1. To select a page, bits D2-D3 determine bits D2-D3 of the page number. To determine which page is selected, bits D0-D1 determine bits D2-D3 of currently selected page number. This is illustrated in Figure 15-2 on page 373.

## Index FFh - The 16 Bit Interface Control Register

This register actually contains a miscellaneous group of control bits:

D7 - 16-bit Bus Status (read only) - indicates if the VGA is installed in a 16-bit slot
D6,D5 - Pointer Bank Select - which bank of memory the graphics cursor pattern is stored in
D4 - Enable access to display beyond 256K
D3 - 16-bit ROM interface enable
D2 - Fast Write Enable
D1 - 16-bit I/O Enable - enables the VGA for 16-bit wide processor I/O operations.
D0 - 16-bit Memory Enable - enables the VGA for 16-bit wide processor memory operations.

A board that is enabled for 16-bit I/O or memory operations will still properly handle all 8-bit wide processor operations and will still operate correctly if installed in an 8-bit wide slot.

# The BIOS

## Interrupt Vectors Used by the BIOS

INT 2, the system NMI vector, is used by the BIOS while in CGA or Hercules emulation mode to interrupt the processor after certain types of I/O operations so that the emulation firmware can properly maintain the state of the VGA display circuitry. INT 10h is the normal vector used to access video BIOS functions. INT 42h is used to retain the motherboard BIOS video services vector so that it can be used to service a secondary video adapter if necessary. INT 43h points to a secondary character generator in CGA graphics modes. INT 1Dh is used in emulation modes to point to a table of parameters for the CRT Controller. INT 1Fh points to a secondary character generator in display modes 4, 5 and 6.

## Added BIOS Functions

The extended modes cannot be selected using Mode Select service of BIOS (function 0). Instead a new extended function must be used.

Included in the VGA1024i BIOS are a number of new functions that are specific to Video Seven boards. These new functions are collectively grouped as BIOS function 6Fh:

Sub function 0 - Inquire
Sub function 1 - Get Info
Sub function 4 - Get Mode and Screen Resolution
Sub function 5 - Extended Set Mode
Sub function 6 - Select Autoswitch Mode
Sub function 7 - Get Video Memory Configuration

### *Inquire (Sub function 0)*

**Input Parameters:**

AH = 6Fh
AL = 0

**Return Value:**

BX = ASCII 'V7' if BIOS extensions are present

## Get Info (Sub function 1)

**Input Parameters:**

AH = 6Fh
AL = 1

**Return Value:**

AH = status
D7,D6 = Diagnostic bits
D5 = Display type (0 = color, 1 = monochrome)
D4 = Display resolution (0 = >200 lines)
D3 = Vertical sync
D2 = Light pen switch
D1 = Light pen flip-flop
D0 = Display enable (0 = enabled)

## Get Mode and Screen Resolution (Sub function 4)

**Input Parameters:**

AH = 6Fh
AL = 4

**Return Value:**

AL = current display mode
BX = Horizontal text columns or graphics pixels
CX = Vertical text rows or graphics pixels

## Extended Set Mode (Sub function 5)

**Input Parameters:**

AH = 6Fh
AL = 5

**Return Value:**

None.

## *Select Autoswitch Mode (Sub function 6)*

**Input Parameters:**

AH = 6Fh
AL = 6
BL = Autoswitch mode select
    00 = EGA/VGA modes only
    01 = VGA/EGA/CGA/MGA modes
    02 = 'Boot-up' CGA/MGA modes
BH = Enable/Disable (0 = enable, 1 = disable)

## *Get Video Memory Configuration (Sub function 7)*

**Input Parameters:**

AH = 6Fh
AL = 7

**Return Value:**

AL = 6Fh
AH = Memory size
    D7 = 1 if adapter uses VRAM
    D6-D0 = Number of 256K blocks of display memory
BX = Chip revision
CX = 0

# Programming Examples

## Display Memory Paging

The display memory paging mechanism of the VGA1024i maps selected portions of the display memory to the processor. Operation of display memory paging is very similar to the paging mechanism used for expanded memory boards (also called EMS or LIM memory). A 64K or 128K logical page of VGA RAM (a chunk of display memory) is mapped into the PC host address space in the normal VGA display memory address space. Paging on Video Seven boards is somewhat convoluted compared to many SuperVGAs. The paging mechanism varies from mode to mode, and often involves more than one I/O address. Memory paging for VGA1024i is illustrated in Figure 15-2.

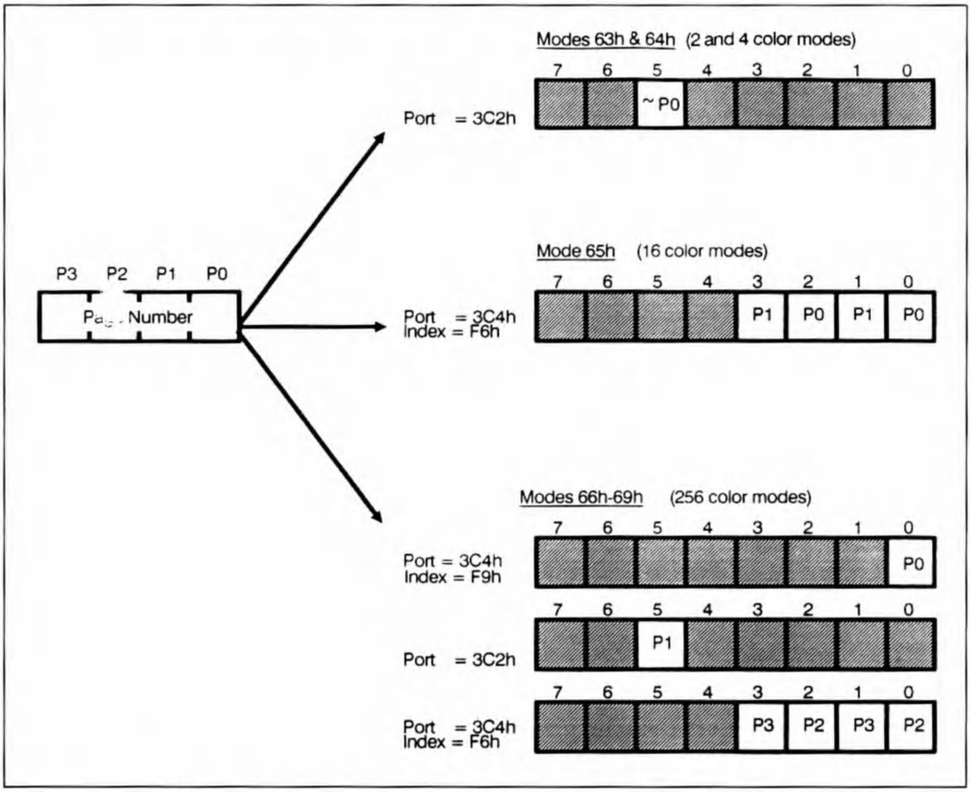


Figure 15-2. Memory paging registers

## Listing 15-1. File: HEADLAND\SELECT.ASM

```
;***********************************************************************
;* File:          SELECT.ASM                                           *
;* Description: This module contains procedures to select mode and to  *
;*              select pages.  It also initializes global variables    *
;*              according to the values in the MODE.INC include file. *
;* Entry Points:                                                       *
;*              _Select_Graphics   - Select a graphics mode            *
;*              _Select_Text       - Set VGA adapter into text mode    *
;*              _Select_Page       - Set read and write page           *
;* Uses:                                                               *
;*              MODE.INC           - Mode dependent constants          *
;*              Following are modes and paths for Video 7 boards:      *
;*        |----- 256 colors ------|  |-- 16 colors --| 4 colors 2 colors *
;*        640x400  640x480  800x600  800x600  1024x768 1024x768 1024x768 *
;* Mode:   66h      67h      69h     6Ah(62h)   65h      64h    63h     *
;* Path: 256COL   256COL   256COL    16COL     16COL    4COL   2COL     *
;***********************************************************************

        INCLUDE VGA.INC
        INCLUDE MODE.INC        ;Mode dependent constants

        PUBLIC  _Select_Graphics
        PUBLIC  _Select_Text
        PUBLIC  _Select_Page
        PUBLIC  _Select_Read_Page
        PUBLIC  _Select_Write_Page

        PUBLIC  Select_Page
        PUBLIC  Select_Read_Page
        PUBLIC  Select_Write_Page
        PUBLIC  Enable_Dual_Page
        PUBLIC  Disable_Dual_Page

        PUBLIC  Graf_Seg
        PUBLIC  Video_Height
        PUBLIC  Video_Width
        PUBLIC  Video_Pitch
        PUBLIC  Video_Pages
        PUBLIC  Video_Colors
        PUBLIC  Ras_Buffer
        PUBLIC  Two_Pages

        PUBLIC  Last_Byte

;-----------------------------------------------------------------------
; Data segment variables
;-----------------------------------------------------------------------

;_DATA   SEGMENT WORD PUBLIC 'DATA'
;_DATA   ENDS

;-----------------------------------------------------------------------
; Constant definitions
;-----------------------------------------------------------------------

;-----------------------------------------------------------------------
; Code segment variables
;-----------------------------------------------------------------------

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Graf_Seg        DW      0A000h          ;Graphics segment addresses
                DW      0A000h
OffScreen_Seg   DW      0A000h          ;First byte beyond visible screen
Video_Pitch     DW      SCREEN_PITCH    ;Number of bytes in one raster
Video_Height    DW      SCREEN_HEIGHT   ;Number of rasters
Video_Width     DW      SCREEN_WIDTH    ;Number of pixels in a raster
Video_Pages     DW      SCREEN_PAGES    ;Number of pages in the screen
```

```
Video_Colors    DW      SCREEN_COLORS    ;Number of colors in this mode
Ras_Buffer      DB      1024 DUP (0)     ;Working buffer
R_Page          DB      0FFh             ;Most recently selected page
W_Page          DB      0FFh
RW_Page         DB      0FFh
Two_Pages       DB      0                ;Indicate separate R & W capability

];*********************************************************************
;*                                                                   *
;* _Select_Graphics(HorizPtr, VertPtr, ColorsPtr)                    *
;*      Initialize VGA adapter to 640x400 mode with                  *
;*      256 colors.                                                  *
;*                                                                   *
;* Entry:                                                            *
;*      None                                                         *
;*                                                                   *
;* Returns:                                                          *
;*      VertPtr   - Vertical resolution                              *
;*      HorizPtr  - Horizontal resolution                            *
;*      ColorsPtr - Number of supported colors                       *
;*                                                                   *
;*********************************************************************

Arg_HorizPtr    EQU     WORD PTR [BP+4] ;Formal parameters
Arg_VertPtr     EQU     WORD PTR [BP+6] ;Formal parameters
Arg_ColorsPtr   EQU     WORD PTR [BP+8] ;Formal parameters

_Select_Graphics PROC NEAR
        PUSH    BP                       ;Standard C entry point
        MOV     BP,SP

        PUSH    DI                       ;Preserve segment registers
        PUSH    SI
        PUSH    DS
        PUSH    ES

        ; Select graphics mode

        MOV     AX,6F05h                 ;Select graphics mode
        MOV     BX,GRAPHICS_MODE
        INT     10h
        MOV     DX,3C4h                  ;Enable extended register bank access
        MOV     AX,0EA06h
        OUT     DX,AX

        ; Enable access to second bank of 256k (mode 66h does not do this)

IFE     (GRAPHICS_MODE-66h)
        MOV     DX,3C4h                  ;Fetch address of extended registers
        MOV     AL,0FFh                  ;Fetch index of control register
        OUT     DX,AL                    ;Select control registers
        INC     DX ·                     ;Point to data
        IN      AL,DX                    ;Read the old value
        OR      AL,10h                   ;Force bit4 to zero
        OUT     DX,AL
ENDIF

        ; Reset 'last selected page'

        MOV     AL,0FFh                  ;Use 'non-existent' page number
        MOV     CS:R_Page,AL             ;Set currently selected page
        MOV     CS:W_Page,AL
        MOV     CS:RW_Page,AL

        ; Set return parameters

        MOV     SI,Arg_VertPtr           ;Fetch pointer to vertical resolution
        MOV     WORD PTR [SI],SCREEN_HEIGHT     ;Set vertical resolution
        MOV     SI,Arg_HorizPtr          ;Fetch pointer to horizontal resolution
        MOV     WORD PTR [SI],SCREEN_WIDTH      ;Set horizontal resolution
        MOV     SI,Arg_ColorsPtr         ;Fetch pointer to number of colors
```

```
        MOV     WORD PTR [SI],SCREEN_COLORS     ;Set number of colors

        ; Clean up and return to caller

        POP     ES                      ;Restore segment registers
        POP     DS
        POP     SI
        POP     DI

        MOV     SP,BP                   ;Standard C exit point
        POP     BP
        RET
_Select_Graphics ENDP

;*************************************************************************
;                                                                        *
; Select_Page                                                            *
; Entry:                                                                 *
;       AL - Page number                                                 *
;                                                                        *
;*************************************************************************

Select_Page     PROC NEAR
        CMP     AL,CS:RW_Page           ;Check if already selected
        JNE     SP_Go
        RET
SP_Go:
        PUSH    AX
        PUSH    BX
        PUSH    DX

        MOV     CS:RW_Page,AL           ;Save most recently selected page
        MOV     CS:R_Page,0FFh
        MOV     CS:W_Page,0FFh
        MOV     AH,AL                   ;Copy page number for later

        ;****************************************************************
        ; Page select for 256 color modes
        ;       0  3C4.F6.0
        ;       1  3C2.  .5
        ;       2  3C4.F6.0&2
        ;       3  3C4.F6.1&3
        ;****************************************************************

IFE (SCREEN_COLORS - 256)
        MOV     DX,3CCh                 ;Fetch value of Misc Input Reg
        IN      AL,DX
        AND     AL,NOT 20h              ;Move bit1 from PageNo into bit5 of
        AND     AH,2                    ;   Misc Output Register
        SHL     AH,1
        SHL     AH,1
        SHL     AH,1
        SHL     AH,1
        OR      AL,AH
        MOV     DX,3C2h
        OUT     DX,AL

        MOV     DX,3C4h                 ;Move bit0 from PageNo into bit0 of
        MOV     AL,0F9h                 ;Sequencer extension reg F9
        MOV     AH,CS:RW_Page
        AND     AH,1
        OUT     DX,AX

        MOV     AL,0F6h                 ;Move bit2 from PageNo into bit0 & bit2
        OUT     DX,AL                   ;and  bit3             into bit1 & bit3
        INC     DX                      ;of Sequencer extension reg F6
        IN      AL,DX
        AND     AL,0F0h

        MOV     BL,CS:RW_Page           ;...isolate bit2 & 3
        AND     BL,0Ch
```

```
        MOV     BH,BL
        SHR     BL,1                        ;...copy bit3&2 into bit1&0
        SHR     BL,1
        OR      AL,BH                       ;...add into AL
        OR      AL,BL
        OUT     DX,AL
ENDIF

        ;*****************************************************************
        ; Page select for 16 color modes
        ;       0  3C4.F6.0&2
        ;       1  3C4.F6.1&3
        ;*****************************************************************

IFE (SCREEN_COLORS - 16)
        MOV     DX,3C4h                     ;Address of extended bank
        MOV     AL,0F6h                     ;Move bit0 from PageNo into bit0 & bit2
        OUT     DX,AL                       ;and  bit1              into bit1 & bit3
        INC     DX                          ;of Sequencer extension reg F6
        IN      AL,DX
        AND     AL,0F0h

        MOV     BL,CS:RW_Page               ;...isolate bit0 & 31
        AND     BL,03h
        MOV     BH,BL
        SHL     BL,1                        ;...copy bit0&1 into bit2&3
        SHL     BL,1
        OR      AL,BH                       ;...add into AL
        OR      AL,BL
        OUT     DX,AL
ENDIF

        ;*****************************************************************
        ; Page select for 4 and 2 color modes
        ;       0 3CC.5
        ;*****************************************************************

IF  SCREEN_COLORS LT 16
        MOV     DX,3CCh                     ;Fetch value of Misc Input Reg
        IN      AL,DX
        AND     AL,NOT 20h                  ;Move ~bit0 from PageNo into bit5 of
        NOT     AH                          ;   Misc Output Register
        AND     AH,1
        SHL     AH,1
        SHL     AH,1
        SHL     AH,1
        SHL     AH,1
        SHL     AH,1
        OR      AL,AH
        MOV     DX,3C2h
        OUT     DX,AL
ENDIF
        POP     DX
        POP     BX
        POP     AX
        RET
Select_Page     ENDP

;*************************************************************************
;                                                                        *
; Select_Read_Page                                                       *
;       There is no separate Read/Write Page capability                  *
; Entry:                                                                 *
;       AL - Page number                                                 *
;                                                                        *
;*************************************************************************

Select_Read_Page PROC NEAR
        RET
Select_Read_Page ENDP
```

```
;*********************************************************************
;                                                                    *
; Select_Write_Page                                                  *
;       There is no separate Read/Write Page capability             *
; Entry:                                                             *
;       AL - Page number                                             *
;                                                                    *
;*********************************************************************

Select_Write_Page PROC NEAR
        RET
Select_Write_Page ENDP

;*********************************************************************
;                                                                    *
; _Select_Page(PageNumber)                                           *
; Entry:                                                             *
;       PageNumber - Page number                                     *
;                                                                    *
;*********************************************************************

Arg_PageNumber   EQU     BYTE PTR [BP+4]

_Select_Page    PROC NEAR
        PUSH    BP                      ;Setup frame pointer
        MOV     SP,BP
        MOV     AL,Arg_PageNumber       ;Fetch argument
        POP     BP                      ;Restore BP
        JMP     Select_Page
_Select_Page    ENDP

;*********************************************************************
;                                                                    *
; _Select_Read_Page(PageNumber)                                      *
; Entry:                                                             *
;       PageNumber- Page number for read                             *
;                                                                    *
;*********************************************************************

Arg_PageNumber  EQU     BYTE PTR [BP+4]

_Select_Read_Page       PROC NEAR
        PUSH    BP                      ;Setup frame pointer
        MOV     SP,BP
        MOV     AL,Arg_PageNumber       ;Fetch argument
        POP     BP                      ;Restore BP
        JMP     Select_Read_Page
_Select_Read_Page   ENDP

;*********************************************************************
;                                                                    *
; _Select_Write_Page(PageNumber)                                     *
; Entry:                                                             *
;       PageNumber - Page number for write                           *
;                                                                    *
;*********************************************************************

Arg_PageNumber  EQU     BYTE PTR [BP+4]

_Select_Write_Page      PROC NEAR
        PUSH    BP                      ;Setup frame pointer
        MOV     SP,BP
        MOV     AL,Arg_PageNumber       ;Fetch argument
        POP     BP                      ;Restore BP
        JMP     Select_Write_Page
_Select_Write_Page  ENDP
```

```
;***********************************************************************
;*                                                                     *
;* _Select_Text                                                        *
;*      Set VGA adapter to text mode                                   *
;*                                                                     *
;***********************************************************************

_Select_Text    PROC NEAR
        MOV     AX,TEXT_MODE            ;Select mode 3
        INT     10h                     ;Use BIOS to reset mode
        RET
_Select_Text    ENDP

;***********************************************************************
;                                                                      *
; Enable_Dual_Page                                                     *
; Disable_Dual_Page                                                    *
;                                                                      *
; Entry:                                                               *
;       AL - Page number                                               *
;                                                                      *
;***********************************************************************

Enable_Dual_Page PROC NEAR
        RET
Enable_Dual_Page ENDP

Disable_Dual_Page PROC NEAR
        RET
Disable_Dual_Page ENDP

Last_Byte:
_Text   ENDS
        END
```

## Graphics Cursor Control

The VGA1024i includes hardware support for a graphics cursor that can significantly reduce the processor overhead required for cursor control. Its usefulness is limited, however, since the hardware cursor cannot be used in 256-color modes. Figure 15-3 illustrates the operation of the hardware graphics cursor. Seven registers in the extended register bank are involved in the definition and control of the graphics cursor.

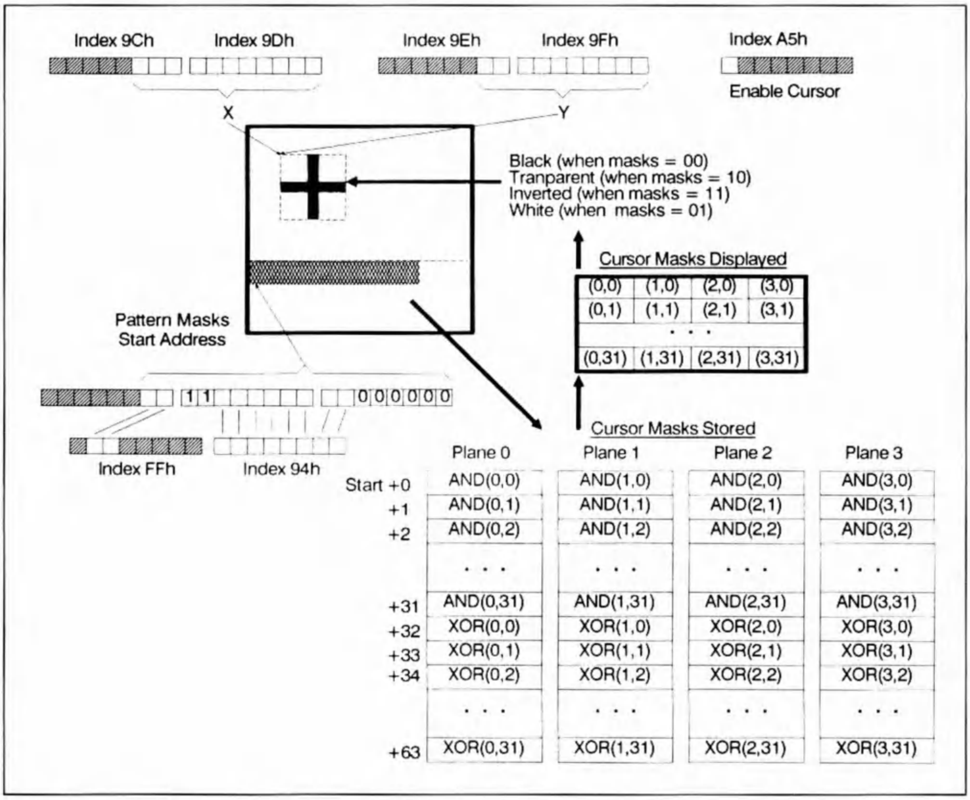


**Figure 15-3. Hardware cursor registers**

Hardware cursors operate differently than software cursors. Since the cursor is drawn as an overlay on the screen, there is never any need to save background data in the cursor area. The cursor is defined by two monochrome bitmaps, or masks, which correspond to the conventional AND and XOR masks used for software cursors (for more on software cursors see our previous text, *Programmer's Guide to the EGA/VGA*).

Cursor pattern data must be loaded into off-screen display memory in a scrambled format. Figure 15-3 shows cursor pattern locations. Each row of cursor, for each mask, is defined by four bytes of pattern (32 bits for each 32-pixel row of the cursor), each byte in a separate plane. Each byte defines 8 pixels, with the most significant bit corresponding to left-most pixel. Bytes for AND mask are in the first 32 bytes (of each plane), and for XOR mask in next 32 bytes (of each plane). Each byte in Figure 15-3 is labeled as (column, row) to indicate which byte in the cursor it controls.

The programming example in Listing 15-2 illustrates how to define cursor shape and how to move the cursor around the screen. Three procedures are provided. Set_Cursor is used to store AND and XOR masks into off-screen display memory, and how to enable the cursor display. Move_Cursor is used to determine where the cursor is displayed. Remove_Cursor disables the cursor display.

### Listing 15-2. File: HEADLAND\HWCURSOR.ASM

```
;***********************************************************************
;*                                                                     *
;* File:        HWCURSOR.ASM                                           *
;* Description: This module contains procedures to demonstrate use of a *
;*              hardware cursor.  It defines cursor shape, moves       *
;*              cursor around the screen, and removes cursor.          *
;*                                                                     *
;* Entry Points:                                                       *
;*                                                                     *
;*              _Set_Cursor                                            *
;*              _Move_Cursor                                           *
;*              _Remove_Cursor                                         *
;*                                                                     *
;* Uses:                                                               *
;*              _Select_Page                                           *
;*              _Graf_Seg                                              *
;*              _Video_Height                                          *
;*              _Video_Pitch                                           *
;*                                                                     *
;***********************************************************************

        INCLUDE VGA.INC
        INCLUDE MODE.INC        ;Mode dependent constants

        EXTRN   Graf_Seg:WORD
        EXTRN   Video_Pitch:WORD
        EXTRN   Video_Height:WORD
        EXTRN   Video_Colors:WORD
        EXTRN   _BitBlt:NEAR
        EXTRN   Select_Page:NEAR

        PUBLIC  _Set_Cursor
        PUBLIC  _Move_Cursor
        PUBLIC  _Remove_Cursor

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

;-----------------------------------------------------------------------
; Common cursor definitions
;-----------------------------------------------------------------------
```

```
;**********************************************************************
;*                                                                    *
;* _Set_Cursor(AND_Mask, XOR_Mask, FG_Color, BG_Color)               *
;*      This procedure saves the two masks in the offscreen memory   *
;*      according to Video 7 schema. Colors are ignored.             *
;*                                                                    *
;* Entry:                                                             *
;*      AND_Mask - 4x32 bytes with AND mask                           *
;*      XOR_Mask - 4x32 bytes with XOR mask                           *
;*      BG_Color - Foreground color                                   *
;*      FG_Color - Background color                                   *
;*                                                                    *
;**********************************************************************

Arg_AND_Mask    EQU     WORD PTR [BP+4] ;Formal parameters
Arg_XOR_Mask    EQU     WORD PTR [BP+6]
Arg_BG_Color    EQU     BYTE PTR [BP+8]
Arg_FG_Color    EQU     BYTE PTR [BP+10]

_Set_Cursor     PROC NEAR
        ; Jump to software cursor routines if 256-color mode

        CMP    WORD PTR CS:Video_Colors,256
        JNE     Set_HW_Cursor
        JMP     Set_SW_Cursor
Set_HW_Cursor:

        ; Save registers

        PUSH    BP                      ;Standard high-level entry
        MOV     BP,SP

        PUSH    SI                      ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Enable planar write to display memory
        ; This needs to be done so that cursor masks can be loaded
        ; one plane at a time, one row of mask per addressable byte.

        MOV     DX,GRAPHICS_CTRL_PORT   ;Address of graphics controller
        MOV     AL,MISC_REG
        OUT     DX,AL                   ;Select misc register
        INC     DX
        IN      AL,DX                   ;Read misc reg value
        PUSH    AX                      ;Save to be restored when done
        AND     AL,01h                  ;Set no chain, memory at A000
        OR      AL,04h
        OUT     DX,AL
        DEC     DX

        MOV     AL,MODE_REG             ;Select mode registers
        OUT     DX,AL
        INC     DX
        IN      AL,DX                   ;Read mode reg value
        PUSH    AX                      ;Save for later
        MOV     AL,01h                  ;Disable odd/even, select latch write
        OUT     DX,AL

        MOV     DX,SEQUENCER_PORT       ;Address of Sequencer
        MOV     AL,4                    ;Select memory mode reg
        OUT     DX,AL
        INC     DX
        IN      AL,DX                   ;Read memory mode reg
        PUSH    AX                      ;Save value for later
        MOV     AL,6                    ;Disable odd/even and double odd/even
        OUT     DX,AL
        DEC     DX
```

```
        MOV     AL,2                    ;Select plane enable
        OUT     DX,AL
        INC     DX
        IN      AL,DX                   ;Read plane enable register
        PUSH    AX                      ;Save for later
        MOV     AL,0Fh                  ;Enable all planes for write
        OUT     DX,AL

        ; Select page where the masks will be stored
        ; (last 64 bytes of each plane in last on-screen page)

        MOV     AX,CS:Video_Height      ;Compute page number of last line
        MUL     CS:Video_Pitch
        MOV     AL,DL                   ;Select the page
        CALL    Select_Page

        ; Set page number and offset for cursor mask location

        ROR     AL,1                    ;Copy page number into bits 5&6
        ROR     AL,1
        ROR     AL,1
        MOV     AH,AL
        MOV     DX,3C4h                 ;Address of extended register bank
        MOV     AL,0FFh                 ;Address if misc reg
        OUT     DX,AL                   ;Select misc reg
        INC     DX
        IN      AL,DX                   ;Read current value of misc reg
        AND     AL,NOT 60h              ;Clear bits 5&6
        OR      AL,AH                   ;Move page number into bits 5&6
        OUT     DX,AL                   ;Set page number for cursor
        DEC     DX

        MOV     DX,3C4h                 ;Address of extended bank
        MOV     AL,94h                  ;Index of pointer pattern address reg
        MOV     AH,0FFh                 ;Indicate last pointer
        OUT     DX,AX                   ;Set the new address

        ; Copy masks to off-screen memory

        MOV     ES,CS:Graf_Seg          ;Segment of display memory
        MOV     DI,-64                  ;Offset is 64 bytes before end of page
        MOV     SI,Arg_AND_Mask         ;Address of AND mask
        MOV     AL,0F2h                 ;Reset latch index
        OUT     DX,AL
        INC     DX
        IN      AL,DX

        MOV     CX,32                   ;Initialize counter
Copy_AND_Loop:
        LODSB                           ;Fetch next value of AND mask
        OUT     DX,AL                   ;Load next latch
        LODSB                           ;Fetch next value of AND mask
        OUT     DX,AL                   ;Load next latch
        LODSB                           ;Fetch next value of AND mask
        OUT     DX,AL                   ;Load next latch
        LODSB                           ;Fetch next value of AND mask
        OUT     DX,AL                   ;Load next latch
        STOSB                           ;Write latches into display memory
        LOOP    Copy_AND_Loop

        MOV     CX,32                   ;Initialize counter
        MOV     SI,Arg_XOR_Mask         ;Fetch pointer to XOR mask
Copy_XOR_Loop:
        LODSB                           ;Fetch next value of mask
        OUT     DX,AL                   ;Load next latch
        LODSB                           ;Fetch next value of mask
        OUT     DX,AL                   ;Load next latch
        LODSB                           ;Fetch next value of mask
        OUT     DX,AL                   ;Load next latch
        LODSB                           ;Fetch next value of mask
        OUT     DX,AL                   ;Load next latch
```

```
        STOSB                           ;Write latches into display memory
        LOOP    Copy_XOR_Loop

        ; Set cursor postion at x=0 and y=last_line+1

        MOV     DX,3C4h                 ;Address of extended registers
        MOV     AL,9Ch                  ;Index of cursor x
        XOR     AH,AH                   ;Value
        OUT     DX,AX                   ;Set hi-x to 0
        INC     AL
        OUT     DX,AX                   ;Set lo-x to 0
        INC     AL
        MOV     BX,CS:Video_Height      ;Fetch number of last_line+1
        MOV     AH,BH
        OUT     DX,AX                   ;Set hi-y
        INC     AL
        MOV     AH,BL
        OUT     DX,AX                   ;Set lo-y

        ; Enable the cursor (will be below last on-screen line)

        MOV     DX,3C4h                 ;Address of extended registers
        MOV     AL,0A5h                 ;Index of cursor attr reg
        OUT     DX,AL                   ;Select cursor attr reg
        INC     DX
        IN      AL,DX                   ;Fetch current value
        OR      AL,80h                  ;Turn cursor on
        OUT     DX,AL

        ; Restore to original mode

        MOV     DX,SEQUENCER_PORT       ;Sequencer address
        MOV     AL,2                    ;Select plane select reg
        OUT     DX,AL
        INC     DX
        POP     AX                      ;Restore original value
        OUT     DX,AL
        DEC     DX

        MOV     AL,4                    ;Select mode reg
        OUT     DX,AL
        INC     DX
        POP     AX                      ;Restore original value
        OUT     DX,AL

        MOV     DX,GRAPHICS_CTRL_PORT   ;Address of Graphics Controller
        MOV     AL,MODE_REG             ;Select mode reg
        OUT     DX,AL
        INC     DX
        POP     AX                      ;Restore original value
        OUT     DX,AL
        DEC     DX

        MOV     AL,MISC_REG             ;Select misc reg
        OUT     DX,AL
        INC     DX
        POP     AX                      ;Restore original value
        OUT     DX,AL

        ; Clean up and return

        POP     DS                      ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI

        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
_Set_Cursor     ENDP
```

```
;**********************************************************************
;*                                                                    *
;* _Move_Cursor(Curs_X, Curs_Y)                                       *
;*      This procedure is used to move the cursor from one            *
;*      location to another, by setting new cursor position registers. *
;*                                                                    *
;**********************************************************************

Arg_Curs_X      EQU     WORD PTR [BP+4] ;Formal parameters
Arg_Curs_Y      EQU     WORD PTR [BP+6]

_Move_Cursor    PROC    NEAR
        ; Jump to software cursor routines if 256-color mode

        CMP    WORD PTR CS:Video_Colors,256
        JNE     Move_HW_Cursor
        JMP     Move_SW_Cursor
Move_HW_Cursor:

        ; Save registers

        PUSH    BP                      ;Standard high-level entry
        MOV     BP,SP
        SUB     SP,4

        PUSH    SI                      ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Set cursor position

        MOV     DX,3C4h                 ;Address of extended registers
        MOV     AL,9Ch                  ;Index of first cursor pos reg
        MOV     BX,Arg_Curs_x           ;Fetch cursor x

        MOV     AH,BH                   ;Set hi-x
        OUT     DX,AX
        INC     AL

        MOV     AH,BL                   ;Set lo-x
        OUT     DX,AX
        INC     AL

        MOV     BX,Arg_Curs_y           ;Fetch cursor y
        MOV     AH,BH                   ;Set hi-y
        OUT     DX,AX
        INC     AL

        MOV     AH,BL                   ;Set lo-y
        OUT     DX,AX

        ; Clean up and return

        POP     DS                      ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI

        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
_Move_Cursor    ENDP

;**********************************************************************
;*                                                                    *
;* _Remove_Cursor                                                     *
;*      This procedure is used to remove the cursor from the screen   *
;*      by disabling cursor display.                                  *
;*                                                                    *
;**********************************************************************
```

```
_Remove_Cursor  PROC NEAR
        ; Jump to software cursor routines if 256-color mode

        CMP     WORD PTR CS:Video_Colors,256
        JNE      Remove_HW_Cursor
        JMP      Remove_SW_Cursor
Remove_HW_Cursor:

        ; Save registers

        PUSH    BP                      ;Standard high-level entry
        MOV     BP,SP

        PUSH    SI                      ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Disable the cursor

        MOV     DX,3C4h                 ;Address of extended registers
        MOV     AL,0A5h                 ;Index of cursor attr reg
        OUT     DX,AL                   ;Select cursor attr reg
        INC     DX
        IN      AL,DX                   ;Fetch current value
        AND     AL,NOT 80h              ;Turn cursor off
        OUT     DX,AL

        ; Clean up and return

        POP     DS                      ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI

        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
_Remove_Cursor  ENDP

;------------------------------------------------------------------------
;--------------- Software Cursor Routines -----------------------------
;------------------------------------------------------------------------

;------------------------------------------------------------------------
; Common cursor definitions
;------------------------------------------------------------------------

CUR_WIDTH       EQU     32
CUR_HEIGHT      EQU     32

AND_OFFSET      EQU     0               ;Save area offsets in off-screen area
XOR_OFFSET      EQU     CUR_WIDTH
CUR_OFFSET      EQU     2*CUR_WIDTH
MIX_OFFSET      EQU     4*CUR_WIDTH

Last_Cursor_x   DW      0               ;Code segment variables
Last_Cursor_y   DW      0
Save_Area_y     DW      0
Save_Offset     DW      0
```

```
;***********************************************************************
;*                                                                     *
;* _Set_Cursor(AND_Mask, XOR_Mask, FG_Color, BG_Color)                 *
;*      This procedure will expand the two cursor masks into           *
;*      color.  Normally the masks should be stored after the          *
;*      last visible scan line (global parameter 'Video_Height',       *
;*      however in this demo, the cursor masks and the 'save buffer'   *
;*      will be stored immediately above the last line.  This is done  *
;*      so that the reader can clearly see the AND mask, the XOR mask, *
;*      and the area under the cursor in 'save buffer'.                *
;*                                                                     *
;* Entry:                                                              *
;*      AND_Mask - 4x32 bytes with AND mask                            *
;*      XOR_Mask - 4x32 bytes with XOR mask                            *
;*      BG_Color - Foreground color                                    *
;*      FG_Color - Background color                                    *
;*                                                                     *
;***********************************************************************

Arg_AND_Mask    EQU     WORD PTR [BP+4] ;Formal parameters
Arg_XOR_Mask    EQU     WORD PTR [BP+6]
Arg_BG_Color    EQU     BYTE PTR [BP+8]
Arg_FG_Color    EQU     BYTE PTR [BP+10]

Set_SW_Cursor   PROC NEAR
        PUSH    BP                      ;Standard high-level entry
        MOV     BP,SP
        SUB     SP,2

        PUSH    SI                      ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Fill with background

        MOV     CX,0                    ;Set x to start of save area
        MOV     AX,CS:Video_Height      ;Set y to below last line on the screen
        ;!!!!!!!!!!!! The next line should be removed    !!!!!!!!!!!!!!!!!!!
        ;!!!!!!!!!!!! if you do not want to see the save !!!!!!!!!!!!!!!!!!!
        ;!!!!!!!!!!!! regions on the screen              !!!!!!!!!!!!!!!!!!!
        MOV     AX,0                    ;Make visible for demo !!!!!!!!!!!!
        MOV     CS:Save_Area_y,AX       ;Save y for other cursor procs
        MUL     CS:Video_Pitch          ;   multiply y by width in bytes
        ADD     AX,CX                   ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits

        MOV     DI,AX                   ;Set DI to save area offset
        MOV     CS:Save_Offset,AX       ;Save offset for later
        MOV     ES,CS:Graf_Seg          ;Set segment to graphics segment
        MOV     AL,DL                   ;Copy page number into AL
        CALL    Select_Page             ;Select page for save area

        MOV     DX,CUR_HEIGHT           ;Number of scanlines to do
        MOV     BX,CS:Video_Pitch       ;Calculate scan-to-scan increment
        SUB     BX,CUR_WIDTH*2

        MOV     AL,Arg_BG_Color         ;Fetch background color
        MOV     AH,AL                   ;Copy color into AH
Fill_Background:
        MOV     CX,CUR_WIDTH            ;Number of words of AND & XOR mask
        REP     STOSW                   ;Fill next row of AND and XOR masks
        ADD     DI,BX                   ;Point to next scanline (assumes in
                                        ;one page!!!).
        DEC     DX                      ;Check if all scanlines done
        JG      Fill_Background         ;Go do next scanline if not done

        ; Change foreground bits for the AND mask save area

        MOV     DL,CUR_HEIGHT           ;Initialize raster counter
        MOV     DH,Arg_FG_Color         ;Fetch foreground color
```

```
        MOV     DI,CS:Save_Offset         ;Get pointer to save area
        MOV     SI,Arg_AND_Mask           ;Fetch pointer to AND-mask section
        ADD     BX,CUR_WIDTH              ;Adjust scan-to-scan increment

Set_AND_FG:
        LODSW                             ;Fetch next 16 bits from the mask
        XCHG    AL,AH                     ;Swap byte to compensate for 80xx mem
        MOV     CX,16                     ;Number of bits to do
AND_Bit_Loop:
        SHL     AX,1                      ;Move next bit into carry
        JNC     AND_Done                  ;Do not change if bit not set
        MOV     ES:[DI],DH                ;Set pixel to fg color if bit set
AND_Done:
        INC     DI                        ;Update pointer
        LOOP    AND_Bit_Loop              ;If not all 16 bits done do next bit
        XOR     BX,8000h                  ;Toggle high bit of BX to check if
        JS      Set_AND_FG                ; both words have been done

        ADD     DI,BX                     ;Point to next scanline
        DEC     DL                        ;Check if all scanlines done
        JG      Set_AND_FG                ;Go do next scanline if not done

        ; Change foreground bits for the XOR mask save area

        MOV     DL,CUR_HEIGHT             ;Initialize raster counter
        MOV     DH,Arg_FG_Color           ;Fetch foreground color
        MOV     DI,CS:Save_Offset         ;Get pointer to save area
        ADD     DI,XOR_OFFSET             ;Advance pointer to XOR-mask section
        MOV     SI,Arg_XOR_Mask           ;Fetch pointer to XOR-mask

Set_XOR_FG:
        LODSW                             ;Fetch next 16 bits from the mask
        XCHG    AL,AH                     ;Swap byte to compensate for 80xx mem
        MOV     CX,16                     ;Number of bits to do
XOR_Bit_Loop:
        SHL     AX,1                      ;Move next bit into carry
        JNC     XOR_Done                  ;Do not change if bit not set
        MOV     ES:[DI],DH                ;Set pixel to fg color if bit set
XOR_Done:
        INC     DI                        ;Update pointer
        LOOP    XOR_Bit_Loop              ;If not all 16 bits done do next bit
        XOR     BX,8000h                  ;Toggle high bit of BX to check if
        JS      Set_XOR_FG                ; both words have been done

        ADD     DI,BX                     ;Point to next scanline
        DEC     DL                        ;Check if all scanlines done
        JG      Set_XOR_FG                ;Go do next scanline if not done

        ; Set 'last cursor' to save area (this is needed for first
        ; call to Move_Cursor procedure, since first thing done in there
        ; is restore area under 'last cursor' position)

        MOV     AX,CS:Save_Area_y         ;Fetch save area y
        MOV     CS:Last_Cursor_y,AX       ;Set last cursor y
        MOV     CS:Last_Cursor_x,CUR_OFFSET  ;Set last cursor x

        ; Clean up and return

        POP     DS                        ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI
        MOV     SP,BP                     ;Restore stack
        POP     BP
        RET
Set_SW_Cursor   ENDP
```

```
;**********************************************************************
;*                                                                    *
;* _Move_Cursor(Curs_X, Curs_Y)                                       *
;*      This procedure is used to move the cursor from one            *
;*      location to another. The cursor move is performed using the   *
;*      following steps:                                              *
;*              1 - Check if new cursor is outside 'cursor block'     *
;*              2 - If outside 'cursor block' restore area under      *
;*                  previous block.                                   *
;*                  Save area under new block.                        *
;*              3 - Copy saved are into cursor build area (both save and*
;*                  build areas are normally off-screen).             *
;*              4 - Combine AND and XOR masks with build area.        *
;*              5 - Copy build area to where new cursor should be (this *
;*                  in most cases overwrites the old cursor).         *
;*      The 'build area' is a rectangle twice the size of the cursor. *
;*      It is used to eliminate flicker for small movement of the     *
;*      cursor, since cursor may not need to be erased if it moves    *
;*      only by a few pixels.                                         *
;*                                                                    *
;* Entry:                                                             *
;*      Curs_X - Position of the new cursor                           *
;*      Curs_Y                                                        *
;*                                                                    *
;**********************************************************************

Arg_Curs_X      EQU     WORD PTR [BP+4] ;Formal parameters
Arg_Curs_Y      EQU     WORD PTR [BP+6]

Curs_X          EQU     WORD PTR [BP-2]
Curs_Y          EQU     WORD PTR [BP-4]

Move_SW_Cursor  PROC    NEAR
        PUSH    BP                      ;Standard high-level entry
        MOV     BP,SP
        SUB     SP,4

        PUSH    SI                      ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Check if new area needs to be saved

        MOV     AX,Arg_Curs_x           ;Fetch new x
        AND     AX,NOT(CUR_WIDTH-1)     ;Round to nearest buffer block
        MOV     BX,Arg_Curs_y           ;Fetch new y
        AND     BX,NOT(CUR_HEIGHT-1)    ;Round to nearest buffer block

        CMP     AX,CS:Last_Cursor_x     ;Check if x moved into next block
        JNE     Cursor_New_Block
        CMP     BX,CS:Last_Cursor_y     ;Check if y moved into next block
        JNE     Cursor_New_Block
        JMP     Build_Cursor

        ; For new block call to remove old cursor, then use _BitBlt
        ; to save block under next cursor location into the save area

Cursor_New_Block:
        CALL    _Remove_Cursor          ;Restore last location
        MOV     AX,Arg_Curs_x           ;Fetch new x
        AND     AX,NOT(CUR_WIDTH-1)     ;Round to nearest buffer block
        MOV     CS:Last_Cursor_x,AX     ;Save as 'last x'
        MOV     AX,Arg_Curs_y           ;Fetch new y
        AND     AX,NOT(CUR_HEIGHT-1)    ;Round to nearest buffer block
        MOV     CS:Last_Cursor_y,AX     ;Save as 'last y'

        MOV     AX,2*CUR_HEIGHT         ;Push width and height
        PUSH    AX
        MOV     AX,2*CUR_WIDTH
        PUSH    AX
```

```
        PUSH    CS:Save_Area_y          ;Push x and y of destination
        MOV     AX,CUR_OFFSET
        PUSH    AX
        PUSH    CS:Last_Cursor_y        ;Push x and y of source
        PUSH    CS:Last_Cursor_x
        CALL    _BitBlt
        ADD     SP,12

        ; Use _BitBlt to copy save area into build area

Build_Cursor:
        MOV     AX,2*CUR_HEIGHT         ;Push width and height
        PUSH    AX
        MOV     AX,2*CUR_WIDTH
        PUSH    AX
        PUSH    CS:Save_Area_y          ;Push x and y of destination
        MOV     AX,MIX_OFFSET
        PUSH    AX
        PUSH    CS:Save_Area_y          ;Push x and y of source
        MOV     AX,CUR_OFFSET
        PUSH    AX
        CALL    _BitBlt
        ADD     SP,12

        ; Mix AND & XOR masks into build area (this will work only if all of
        ; the save area is in the same segment!!!)

        MOV     CX,Arg_Curs_x           ;Fetch x
        AND     CX,CUR_WIDTH-1          ;Keep 'odd' bits
        ADD     CX,MIX_OFFSET           ;Add 'base x' of save area
        MOV     AX,Arg_Curs_y           ;Fetch y
        AND     AX,CUR_HEIGHT-1         ;Keep 'odd' bits
        ADD     AX,CS:Save_Area_y       ;Add 'base y' of save area
        MUL     CS:Video_Pitch          ;   multiply y by width in bytes
        ADD     AX,CX                   ;   add x coordinate to compute offset
        ADC     DX,0                    ;   add overflow to upper 16 bits

        MOV     DI,AX                   ;Save offset
        MOV     AL,DL                   ;Select page
        CALL    Select_Page
        MOV     ES,CS:Graf_Seg          ;Set both segments to video buffer
        MOV     DS,CS:Graf_Seg

        MOV     DL,CUR_HEIGHT           ;Initialize raster counter
        MOV     SI,CS:Save_Offset       ;Get pointer to AND & XOR masks
        MOV     BX,CS:Video_Pitch       ;Compute scan-to-scan increment
        SUB     BX,CUR_WIDTH

Mix_Lines:
        MOV     CX,CUR_WIDTH            ;Fetch cursor width
MixBytes:
        LODSB                           ;Fetch next byte of AND mask
        MOV     AH,[DI]                 ;Fetch next byte of destination
        AND     AL,AH                   ;Combine mask with destination
        MOV     AH,[SI+CUR_WIDTH-1]     ;Fetch next byte of XOR mask
        XOR     AL,AH                   ;Combine with previous result
        STOSB                           ;Place result into destination
        LOOP    Mix_Bytes

        ADD     DI,BX                   ;Point to next scanline
        ADD     SI,BX                   ;Point to next scanline
        DEC     DL                      ;Check if all scanlines done
        JG      Mix_Lines               ;Go do next scanline if not done

        ; Use _BitBlt procedure to copy build area to screen (and erase old
        ; cursor with the new cursor block).

        MOV     AX,2*CUR_HEIGHT         ;Push width and height
        PUSH    AX
        MOV     AX,2*CUR_WIDTH
        PUSH    AX
```

```
        PUSH    CS:Last_Cursor_y        ;Push x and y of destination
        PUSH    CS:Last_Cursor_x
        PUSH    CS:Save_Area_y          ;Push x and y of source
        MOV     AX,MIX_OFFSET
        PUSH    AX
        CALL    _BitBlt
        ADD     SP,12

        ; Clean up and return

        POP     DS                      ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI

        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
Move_SW_Cursor  ENDP

;*************************************************************************
;*                                                                       *
;* _Remove_Cursor                                                        *
;*      This procedure is used to remove the cursor from the screen      *
;*      and to restore the screen to its original appearance             *
;*                                                                       *
;*************************************************************************

Remove_SW_Cursor PROC NEAR
        PUSH    BP                      ;Standard high-level entry
        MOV     BP,SP

        PUSH    SI                      ;Save registers
        PUSH    DI
        PUSH    ES
        PUSH    DS

        ; Use _BitBlt to restore area under the last cursor location

        MOV     AX,2*CUR_HEIGHT         ;Push width and height
        PUSH    AX
        MOV     AX,2*CUR_WIDTH
        PUSH    AX
        PUSH    CS:Last_Cursor_y        ;Push last position of cursor
        PUSH    CS:Last_Cursor_x
        PUSH    CS:Save_Area_y          ;Push x and y of destination
        MOV     AX,CUR_OFFSET
        PUSH    AX
        CALL    _BitBlt
        ADD     SP,12

        ; Clean up and return

        POP     DS                      ;Restore segment registers
        POP     ES
        POP     DI
        POP     SI

        MOV     SP,BP                   ;Restore stack
        POP     BP
        RET
Remove_SW_Cursor ENDP

_TEXT   ENDS
        END
```

## Detection and Identification

Headland Technology recommends that the presence of their BIOS be detected using extended BIOS function 0 (Inquire). The presence of the V7VGA chip can then be detected using extended BIOS function 7 (Get Video Configuration). Code similar to the following can be used to detect the V7VGA chip:

```
        ; Check for Video Seven product using BIOS
        MOV          AX,6F00h                    ;Load BIOS function code
        INT          10h                         ;Invoke BIOS service
        CMP          BX,'V7'                     ;Check for Video Seven board
        JNE          Not_Video7                  ;Quit if not Video Seven board
        ; Check for V7VGA chip
        MOV          AX,6F07h                    ;Load BIOS function code
        INT          10h                         ;Invoke BIOS service
        CMP          BL,70h                      ;Check chip version
        JB           Not_V7VGA                   ;Quit if version below 70h
        CMP          BL,7Fh
        JA           Not_V7VGA                   ;Quit if version above 7Fh
V7VGA_Found:
        ...
```

Headland Technology included a presence detection mechanism in their VGA chips that is simple, reliable, and places no requirements on the BIOS ROM. Two new registers in the extended register bank are used to detect V7VGA presence and revision level.

The following programming example shows how to use the Identification register to implement a presence test that should have no adverse side effects when executed on other types of EGA or VGA display adapters. In the HT-208 (V7VGA) chip, the value written to CRTC register 0Ch is XORed by hardware with EAh and the result is placed in CRTC register 1Fh. Code similar to the following can be used to verify this function, and if verified, confirm presence of V7VGA chip.

```
        ; Preserve CRTC register 0Ch
        MOV          DX,CRTC_ADDRESS             ;Fetch CRTC address (3B4h or 3D4h)
        MOV          AL,0Ch                      ;Index of Start Address
        OUT          DX,AL                       ;Select register
        INC          DX                          ;Address of data
        IN           AL,DX                       ;Read current value
        MOV          AH,AL                       ;Save the value for later
        ; Set CRTC register 0Ch to 0 (will be XORed with EAh and placed in reg 1Fh)
        MOV          AL,0                        ;Value to write
        OUT          DX,AL                       ;Write to register
        DEC          DX
        ; Verify that CRTC register 1Fh contains EAh (EAh was XORed with reg 0Ch)
        MOV          AL,1Fh                      ;Index of Identification register
        OUT          DX,AL                       ;Select ID register
        INC          DX
        IN           AL,DX                       ;Read the ID register
        CMP          AL,0EAh                     ;Is it EAh?
        JNE          Not_V7_Board                ;...No, not a V7 board
        DEC          DX
       ; Set CRTC register 0C to FFh (will be XORed with EAh and placed in reg 1Fh)
       MOV          AL,0Ch                      ;Index of Start Address High
       OUT          DX,AL                       ;Select register
       INC          DX
       MOV          AL,0FFh                     ;Value to write
       OUT          DX,AL                       ;Set Start Address to FFh
       DEC          DX
```

```
    ; Verify that CRTC register 1Fh contains 15h (EAh was XORed with reg 0Ch)
    MOV             AL,1Fh                          ;Index of ID register
    DX,AL                                           ;Select ID register
    INC             DX
    IN              AL,DX                           ;Read the ID register
    CMP             AL,15h                          ;Is it 15h?
    JNE             Not_V7_Board                    ;...No, not a V7 board
    DEC             DX                              ;decrement to index register
    ; Restore CRTC register 0Ch to its original value
    MOV             AL,0Ch                          ;Index of Start Address
    OUT             DX,AX                           ;Restore the initial value
V7_Found:
    ; Read version to distinguish V7VGA and VEGA chips
    ; (assumes that extended registers are enabled)
    MOV             DX,3C4h                         ;Address of extended bank
    MOV             AL,8Eh                          ;Index of version register
    OUT             DX,AL                           ;Select version register
    INC             DX
    IN              AL,DX                           ;Read chip version
    CMP             AL,70h                          ;Check version range
    JB              Not_V7                          ;Out of range
    CMP             AL,80h                          ;Is it in 70h or 7Fh?
    JB              V7VGA_Chip_Found                ;...Yes, found V7VGA chip
VEGA_Chip_Found:                                    ;...No, (80h to FFh) VEGA chip found
    ...
V7VGA_Chip_Found:
```

# 16

# Trident TVGA 8800CS Everex Viewpoint VGA

TRIDENT Microsystems, Incorporated

# Introduction

As with most SuperVGAs, the Everex Viewpoint and all Trident 880CS based adapters are fully IBM VGA-compatible, and also include register level compatibility with EGA, CGA, MDA and Hercules display adapters.

Viewpoint also includes extended high resolution text and graphics modes. High resolution applications software drivers are available for AutoCAD, Autoshade, GEM, Lotus 1-2-3, Symphony, Ventura Publisher, MS-Windows, WordPerfect, and OS/2 Presentation Manager. Additional drivers for Everex products are continually added and are available through the Everex BBS system.

Everex also offers Everex EVGA (EV673), Everex Ultragraphics II VGA (EV236) and Vision Technologies Vision VGA (EV620).

# Chip Versions

Trident VGA chips contain a version number that can be read from a register at I/O address 3C5, index 0Bh. There are currently two versions of the TVGA 8800 chip: 8800BR is version 1, and 8800CS is version 2. The major difference between these two chips is the method used for display memory paging. Version 1 supports only 128K pages; version 2 supports both 128K pages as well as 64K pages.

For the Everex Viewpoint VGA, the version number should always be 2. Unless stated otherwise, all information in this chapter applies to the version 2 chip.

Trident has announced a newer chip, the TVGA 8900, which is capable of operating at a resolution of 1024x768 with 256 colors (using 1 megabyte of RAM). At the time of this writing there are no boards available using this new chip.

# New Display Modes

Table 16-1 lists the enhanced display modes that are supported by the Everex Viewpoint VGA. Enhanced modes are selected via a modified call to the BIOS mode select function (using AX = 0070h and BX = Extended Mode Number). For details, see the section "The BIOS" later in this chapter. Exception is mode 6Ah, which is selected using normal BIOS function 0.

Table 16-1. Enhanced display modes—Everex Viewpoint VGA

| Mode | Type | Resolution | Colors | Memory Required | Display Type |
|---|---|---|---|---|---|
| 03h (1) | Text | 80 col x 34 rows | 16 | 256 KB | VGA |
| 04h (1) | Text | 80 col x 60 rows | 16 | 256 KB | VGA |
| 07h (1) | Text | 100 col x 43 rows | 16 | 256 KB | SuperVGA |
| 08h (1) | Text | 100 col x 75 rows | 16 | 256 KB | SuperVGA |
| 0Ah (1) | Text | 132 col x 25 rows | 16 | 256 KB | EGA |
| 0Bh (1) | Text | 132 col x 44 rows | 16 | 256 KB | EGA |
| 0Ch (1) | Text | 132 col x 25 rows | 16 | 256 KB | CGA |
| 0Eh (1) | Text | 132 col x 25 rows | Mono | 256 KB | MDA |
| 0Fh (1) | Text | 132 col x 44 rows | Mono | 256 KB | MDA |
| 16h (1) | Text | 80 col x 30 rows | 16 | 256 KB | VGA |
| 18h (1) | Text | 100 col x 37 rows | 16 | 256 KB | SuperVGA |
| 40h (1) | Text | 132 col x 30 rows | 16 | 256 KB | VGA |
| 50h (1) | Text | 132 col x 30 rows | Mono | 256 KB | VGA |
| 02h (1) | Graphics | 800x600 | 16 | 256 KB | SuperVGA |
| 14h (1) | Graphics | 640x400 | 256 | 256 KB | SuperVGA |
| 15h (1) | Graphics | 512x480 | 256 | 256 KB | VGA |
| 20h (1) | Graphics | 1024x768 | 16 | 512 KB | 8514 |
| 30h (1) | Graphics | 640x480 | 256 | 512 KB | SuperVGA |
| 31h (1) | Graphics | 800x600 | 256 | 512 KB | SuperVGA |
| 60h (1)(2) | Graphics | 1024x768 | 4 | 256 KB | 8514 |
| 6A | Graphics | 800x600 | 16 | 256K | SuperVGA |

Note (1): These are extended mode numbers, which cannot be selected using the standard BIOS function call. For information on how to select extended modes, see the section "The BIOS" later in this chapter.

Note (2): This mode is not available on some versions of BIOS.

Trident recommends that all manufacturers using 8800CS chip support extended modes and mode numbers as listed in Table 16-2 on page 398. Most boards based on Trident 8800CS chips support these modes.

Table 16-2. Enhanced display modes—Trident

| Mode | Type | Resolution | Colors | Memory Required | Display Type |
|---|---|---|---|---|---|
| 50h | Text | 80 col x 30 rows | 16 | 256 KB | VGA |
| 51h | Text | 80 col x 43 rows | 16 | 256 KB | VGA |
| 52h | Text | 80 col x 60 rows | 16 | 256 KB | VGA |
| 53h | Text | 132 col x 25 rows | 16 | 256 KB | VGA (8x14 characters) |
| 54h | Text | 132 col x 30 rows | 16 | 256 KB | VGA |
| 55h | Text | 132 col x 43 rows | 16 | 256 KB | VGA |
| 56h | Text | 132 col x 60 rows | 16 | 256 KB | VGA (8x8 characters) |
| 57h | Text | 132 col x 25 rows | 16 | 256 KB | VGA (9x14 characters) |
| 58h | Text | 132 col x 30 rows | Mono | 256 KB | VGA |
| 59h | Text | 132 col x 43 rows | Mono | 256 KB | VGA |
| 5Ah | Text | 132 col x 60 rows | 16 | 256 KB | VGA (9x8 characters) |
| 5Bh | Graphics | 800x600 | 16 | 256 KB | SuperVGA |
| 5Ch | Graphics | 640x400 | 256 | 256 KB | SuperVGA |
| 5Dh | Graphics | 640x480 | 256 | 512 KB | SuperVGA |
| 5Fh | Graphics | 1024x768 | 16 | 512 KB | 8514 |

# Memory Organization

For all Viewpoint extended modes, display memory organization is patterned after the organization used in one of the standard IBM VGA modes.

Viewpoint includes a display memory paging mechanism that is needed in some display modes to make the entire display memory accessible to the processor. Display memory paging is described in detail later in this chapter.

## High Resolution Text Modes

These modes utilize memory maps that are similar to those used in standard text modes (modes 0,1,2,3 and 7), except that the number of characters per row, and number of rows is increased. This increases the number of bytes used per screen of text. Display memory is organized as shown in Figure 5-1 (see Chapter 5).

## High Resolution Graphics Modes

### *4-color Graphics Mode 60h - 1024x768*

This mode resembles VGA mode 12h, except that only two planes are used. Memory planes 0 and 2 are used to store bytes that have even host memory addresses; planes 1

and 3 store bytes for odd memory addresses. To learn more about this memory organization see the section "Four Planes" in Chapter 9.

### 16-color Graphics Mode 02h - 800x600

Memory organization for this mode resembles VGA mode 12h (640x480 16-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is shown in Figure 7-1. See Chapter 7 for programming examples.

Only 256K of display memory are required to support this mode; display memory paging is not required.

### 16-color Graphics Mode 20h - 1024x768

Memory organization for this mode resembles VGA mode 12h (640x480 16-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is shown in Figure 7-1. See Chapter 7 for programming examples.

512K of display memory are required to support this mode; display memory paging is required. Default colors are the same as for mode 12h (16-color graphics).

### 256-color Graphics Modes, 14h, 15h, 30h, 31h

These modes, because of their higher resolutions, require larger amounts of display memory which exceed the 64K page size of display memory. The Memory Page Select register in the extended register bank is used to select which memory page can be accessed by the processor.

Display memory organization for these modes resembles VGA mode 13h (320x200 256-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. The memory map for these modes can be seen in Figure 8-1 (see Chapter 8).

Default colors are the same as for mode 13h.

## New Registers

To support enhanced display modes and emulations, the Trident chips contain additional registers not found on the standard VGA. These are listed in Table 16-3 on the following page.

Table 16-3. Extended Registers—Trident 8800CS

| Register Name | Address | Index |
|---|---|---|
| CRTC Module Testing register | 3B4h/3D4 | 1Eh |
| Scratch Pad | 3B4h/3D4h | 1Fh |
| Power Up Mode register 1 | 3C4 | 0Ch |
| Power Up Mode register 2 | 3C4 | 0Fh |
| Hardware Version register | 3C4h | 0Bh |
| Mode Control register 1 | 3C4h | 0Eh |
| Mode Control register 2 | 3C4h | 0D |
| CPU Latch Read Back | 3B4h/3D4h | 22h |
| Attribute State Read Back | 3B4h/3D4h | 24h |
| Attribute Index Read Back | 3B4h/3D4h | 26h |
| Video Enable | 3C3h | |
| Display Adapter Enable | 46E8h | |

Registers used in the programming examples are described in detail below.

## Hardware Version Register (I/O Address 3C5h Index 0Bh)

D7-D4 - Reserved
D3-D0 - Hardware version

Reading this register causes the chip to enter version 2 paging mode. Writing this register causes the chip to enter version 1 paging mode. Programming examples in this chapter assume version 2 paging. For more details on paging see the programming examples.

## Mode Control Register 1 (I/O Address 3C5h Index 0Eh)

D7-D4 - Reserved
D3-D0 - 64K page select

This register is used to select page number in version 2 paging mode. In this mode, bit 1 must be written inverted, but will read back the correct (uninverted) value. For example, page 7 would be selected by writing a value of 5; when read back, a value of 7 would be read.

## Scratch Pad Register (I/O Address 3B4h/3D4h Index 1Fh)

This scratch register is used on Everex Viewpoint boards as a 4-bit scratch as follows:

D3 - 44.9MHz oscillator present

D2 - Analog monitor attached
D1 - Memory size (0: 256K, 1: 512K)
D0 - Paged memory mode in effect

## Processor Latch Read Back Register (I/O Address 3B4h/3D4h Index 22h)

This register can be used to read back the current value of the processor data latch in the Graphics Controller for the color plane that is currently enabled for reading.

## Attribute Controller State Register (I/O Address 3B4h/3D4h Index 24h)

D7 - Attribute Controller State (read-only)
D6-D0 - Reserved

### *Attribute Controller State*

indicates whether the next write operation to the Attribute Controller (I/O address 3C0) will be used as a register index or as register data (0 = index, 1 = data).

## Attribute Controller Index Read Back (I/O Address 3B4h/3D4h Index 26h)

This read-only port can be used to read the current value of the index register internal to the Attribute Controller.

# The BIOS

## Extended Mode Select - Function 0

Extended display modes of the Viewpoint VGA are selected by a modified version of the BIOS mode select function.

**Input Parameters:**

AH = 00h
AL = 70h
BL = Extended mode number

**Return Value:**

None

## Return Emulation Status - Function 70H Sub function 0

**Input Parameters:**

AX = 7000h
BX = 0

**Return Value:**

AL = 70h (if supported)
CL = Display type
    0 = MDA
    1 = CGA
    2 = EGA
    3 = Digital multi-frequency
    4 = VGA
    5 = 8514
    6 = SuperVGA (e.g. NEC 2A)
    7 = Analog multi-frequency

CH - Status

    D6,D7 - Display memory size (256/512/1024/2048K)
    D4 - VGA protect enabled
    D0 - 6845 emulation enabled

DX - Board ID

    D15-D4 - Model number
        678h = Viewpoint (EV678)
        236h = Ultragraphics II (EV236)
        620h = Vision VGA (EV620)
        673h = EVGA (EV673)
    D3-D0 - Revision

DI - BIOS version (e.g., 0100h for version V1.00)

## Set Operating Mode - Function 70H Sub function 1

**Input Parameters:**

AX = 7000h
BX = 1
CH = 0: Disable 6845 emulation, 1: Enable 6845 emulation

**Return Value:**

AL = 70h (if supported)

## VGA Register Protect - Function 70H Sub function 2

**Input Parameters:**

AX = 7000h
BX = 2
CH = 0: Disable protect of CRTC 00 to 07, 10h to 17h, Misc output
1: Enable protect

**Return Value:**

AL = 70h (if supported)

## Enable/Disable Fast Mode - Function 70H Sub function 3

**Input Parameters:**

AX = 7000h
BX = 3
CH = 0: Disable fast mode
1: Enable fast mode (default)

**Return Value:**

AL = 70h (if supported)

## Get Paging Function Pointer - Function 70H Subfunction 4

This function returns a far pointer to a subroutine that can be called to select display memory pages. This function can be used to guarantee compatibility with future Everex products.

**Input Parameters:**

AX = 7000h
BX = 4

**Return Value:**

ES:DI = far pointer to page select routine, which can be called with a page number in DL

Everex recommends that paging be implemented using this function, to ensure compatibility with future Everex VGA products.

## Get Mode Supported Info - Function 70H Sub function 5

**Input Parameters:**

AX = 7000h
BX = 5
CL = Maximum number of modes to get info for
DL = Monitor type to get mode info for
ES:DI = Buffer address
CH = Mode type to get info for
  0: to get all modes
  1: to get mono text modes
  2: to get color text modes
  3: to get 4-color (CGA) graphics modes
  4: to get 1-color (CGA) graphics modes
  5: to get 16-color (planar) graphics modes
  6: to get 256-color graphics modes

**Return Value:**

AL = 70h (if supported)
CL = Total number of modes fitting criteria
CH = Size of each record
ES:DI = Info records

| | |
|---|---|
| BYTE | Mode number (Bit 7 set if extended mode) |
| BYTE | Mode format (same as input parameter CH) |
| BYTE | Info bits |
| | D5 - Monochrome |
| | D4 - Interlaced |
| | D3 - Requires 44.9 MHz oscillator |
| | D2 to D1 - Memory required (256K, 512K, 1024K, 2048K) |

| | |
|---|---|
| | D0 - Paged mode |
| BYTE | Font height (bits 0-4), 9 dot (bit 8) |
| BYTE | Text columns |
| BYTE | Text rows |
| WORD | Number of scan lines |
| BYTE | Color info |
| | D7 to D4 - Reserved |
| | D3 to D0 - Bits per pixel |

## Program Mode Parameters - Function 70H Sub function 6

### Input Parameters:

AX = 7000h
BX = 6
ES:DI = Standard 64-byte parameter table
DS:DX = Extra register table

### Return Value:

AL = 70h (if supported)

## Everex Set Mode - Function 70H Sub function 9

### Input Parameters:

AX = 7000h
BX = 9
CH = Setmode AL
CL = Setmode BL

### Return Value:

AL = 70h (if supported)

# Programming Examples

## Display Memory Paging - Version 1 Mode

This is the only memory paging mode available on the version 1 Trident VGA chip (Everex Viewpoint uses version 2 and later devices, which include an additional paging mode). In this mode, display memory is divided into four 128K pages. A full 128K of host address space is used, which means that other display adapters cannot co-reside in the system when this mode is used.

Only one memory page may be selected at a time. Two register bits are used to select pages; these two bits are located in different registers. Table 16-4 shows how memory pages are selected.

**Table 16-4. Memory paging version 1 mode—Everex Viewpoint**

| I/O Addr 3C5h Index 0Eh Bit D1 | I/O Addr 3C2h (read from 3CCh) Bit D5 | Page Number |
|---|---|---|
| 0 | 1 | Page 0 |
| 0 | 0 | Page 1 |
| 1 | 1 | Page 2 |
| 1 | 0 | Page 3 |

## Display Memory Paging - Version 2 Mode

For version 2 Trident VGA chips, an additional paging mode is supported that divides display memory into eight 64K pages. This paging mode is selected by a read from Hardware Version register (I/O address 3C5h index 0Bh). VGA host memory space should be set to 64K via the Miscellaneous register of the Graphics Controller (Address 3CF, index 6, bits D2 and D3).

Page selection is easier in this mode; the desired page number can simply be output to Mode Control register 1 (I/O address 3C5h, index 0Eh). Note, however, that bit D1 of the register must be complemented before the page number is written, but will read back uncomplemented. This is illustrated in Figure 16-1. Table 16-5 contains value read and written for each of the valid page numbers.

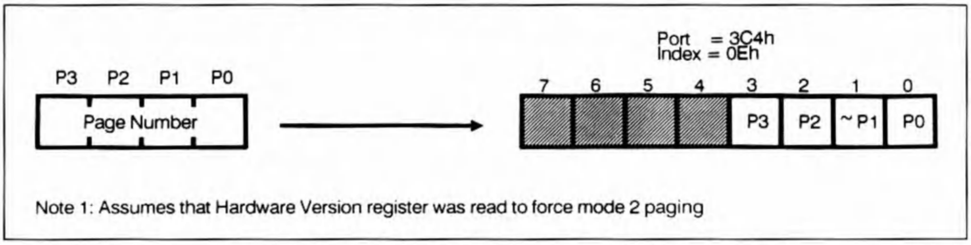


**Figure 16-1. Paging registers—version 2**

Access to Mode Control register 1 is enabled by a read operation from the Hardware Version register (I/O address 3C5h, index 0Bh) and is disabled by a write operation to the Hardware Version register. Access to the page select bits in the Mode Control register 1 should always be prefaced by a read from the Hardware Version register to assure that it is enabled.

**Table 16-5. Memory paging mode 2—Everex Viewpoint**

| | I/O Address 3C5, Index 0Eh | |
|---|---|---|
| | Write Value | Read Value |
| Page Number | D3 D2 D1 D0 | D3 D2 D1 D0 |
| 0 | 0 0 1 0 | 0 0 0 0 |
| 1 | 0 0 1 1 | 0 0 0 1 |
| 2 | 0 0 0 0 | 0 0 1 0 |
| 3 | 0 0 0 1 | 0 0 1 1 |
| 4 | 0 1 1 0 | 0 1 0 0 |
| 5 | 0 1 1 1 | 0 1 0 1 |
| 6 | 0 1 0 0 | 0 1 1 0 |
| 7 | 0 1 0 1 | 0 1 1 1 |

The programming example in Listing 16-1 contains mode select procedures, Select_Graphics and Select_Text, and a paging procedure Select_Page. Select_Graphics contains an example on how to invoke extended modes. Note that in the procedure Select_Graphics, after the mode select, the 64K page is selected using the Miscellaneous register of the Graphics Controller. Version 2 paging is forced by a read from the Hardware Version register. Select_Page contains an example showing how to select a display memory page for version 2 paging mode.

Listing 16-1. File: TRIDENT\SELECT.ASM

```
;***************************************************************************
;* File:         SELECT.ASM                                                *
;* Description: This module contains procedures to select mode and to     *
;*               select pages.  It also initializes global variables       *
;*               according to the values in the MODE.INC include file.    *
;* Entry Points:                                                           *
;*               _Select_Graphics    - Select a graphics mode              *
;*               _Select_Text        - Set VGA adapter into text mode      *
;*               _Select_Page        - Select page for read and write      *
;* Uses:                                                                   *
;*               MODE.INC            - Mode dependent constants            *
;*               Following are EXTENDED modes and paths for Everex boards:*
;*        |----- 256 colors ------|   |-- 16 colors --| 4 colors 2 colors *
;*        640x400  640x480  800x600   800x600  1024x768 1024x768 1024x768 *
;* Mode:   14h      30h      31h        2        20h      60h  N/A        *
;* Path: 256COL   256COL   256COL     16COL     16COL    4COL N/A         *
;***************************************************************************

        INCLUDE VGA.INC
        INCLUDE MODE.INC        ;Mode dependent constants

        PUBLIC  _Select_Graphics
        PUBLIC  _Select_Text
        PUBLIC  _Select_Page
        PUBLIC  _Select_Read_Page
        PUBLIC  _Select_Write_Page

        PUBLIC  Select_Page
        PUBLIC  Select_Read_Page
        PUBLIC  Select_Write_Page
        PUBLIC  Enable_Dual_Page
        PUBLIC  Disable_Dual_Page

        PUBLIC  Graf_Seg
        PUBLIC  Video_Height
        PUBLIC  Video_Width
        PUBLIC  Video_Pitch
        PUBLIC  Video_Pages
        PUBLIC  Ras_Buffer
        PUBLIC  Two_Pages

        PUBLIC  Last_Byte

;-------------------------------------------------------------------------
; Data segment variables
;-------------------------------------------------------------------------

;_DATA   SEGMENT WORD PUBLIC 'DATA'
;_DATA   ENDS

;-------------------------------------------------------------------------
; Constant definitions
;-------------------------------------------------------------------------

EXTEND_REG_ADDR EQU     3C4h            ;IO Address for extended bank registers
VERSION_REG     EQU     00Bh            ;Index for enable/version register
PAGE_REG        EQU     00Eh            ;Index for page register

;-------------------------------------------------------------------------
; Code segment variables
;-------------------------------------------------------------------------

_TEXT   SEGMENT BYTE PUBLIC 'CODE'

Graf_Seg        DW      0A000h          ;Graphics segment addresses
                DW      0B000h
OffScreen_Seg   DW      0A000h          ;First byte beyond visible screen
Video_Pitch     DW      SCREEN_PITCH    ;Number of bytes in one raster
```

```
Video_Height    DW      SCREEN_HEIGHT   ;Number of rasters
Video_Width     DW      SCREEN_WIDTH    ;Number of pixels in a raster
Video_Pages     DW      SCREEN_PAGES    ;Number of pages in the screen
Ras_Buffer      DB      1024 DUP (0)    ;Working buffer
R_Page          DB      0FFh            ;Most recently selected page
W_Page          DB      0FFh
RW_Page         DB      0FFh
Two_Pages       DB      CAN_DO_RW       ;Indicate separate R & W capability

;*************************************************************************
;*                                                                       *
;* _Select_Graphics(HorizPtr, VertPtr, ColorsPtr)                        *
;*      Initialize VGA adapter to 640x400 mode with                      *
;*      256 colors.                                                      *
;*                                                                       *
;* Entry:                                                                *
;*      None                                                             *
;*                                                                       *
;* Returns:                                                              *
;*      VertPtr   - Vertical resolution                                  *
;*      HorizPtr  - Horizontal resolution                                *
;*      ColorsPtr - Number of supported colors                           *
;*                                                                       *
;*************************************************************************

Arg_HorizPtr    EQU     WORD PTR [BP+4] ;Formal parameters
Arg_VertPtr     EQU     WORD PTR [BP+6] ;Formal parameters
Arg_ColorsPtr   EQU     WORD PTR [BP+8] ;Formal parameters

_Select_Graphics PROC NEAR
        PUSH    BP                      ;Standard C entry point
        MOV     BP,SP

        PUSH    DI                      ;Preserve segment registers
        PUSH    SI
        PUSH    DS
        PUSH    ES

        ; Select graphics mode

        MOV     AX,70h                  ;Fn=Select Mode, Mode=Everex Extended
        MOV     BX,GRAPHICS_MODE        ;Set extended mode number
        INT     10h                     ;Use BIOS to select mode

        ; Reset 'last selected page'

        MOV     AL,0FFh                 ;Use 'non-existent' page number
        MOV     CS:R_Page,AL            ;Set currently selected page
        MOV     CS:W_Page,AL
        MOV     CS:RW_Page,AL

        ; Set return parameters

        MOV     SI,Arg_VertPtr          ;Fetch pointer to vertical resolution
        MOV     WORD PTR [SI],SCREEN_HEIGHT     ;Set vertical resolution
        MOV     SI,Arg_HorizPtr         ;Fetch pointer to horizontal resolution
        MOV     WORD PTR [SI],SCREEN_WIDTH      ;Set horizontal resolution
        MOV     SI,Arg_ColorsPtr        ;Fetch pointer to number of colors
        MOV     WORD PTR [SI],SCREEN_COLORS     ;Set number of colors

        ; Enable extended register access for version 2
        MOV     DX,EXTEND_REG_ADDR      ;Address of extended reg bank
        MOV     AL,VERSION_REG          ;Index of version (and enable) reg
        OUT     DX,AL                   ;Select register
        INC     DX                      ;Advance to data port
        IN      AL,DX                   ;Read version to enable version 2 mode

        MOV     DX,GRAPHICS_CTRL_PORT   ;Address of graphics controller
        MOV     AL,MISC_REG             ;Index of miscellaneous register
        OUT     DX,AL                   ;Select misc register
        INC     DX                      ;Advance to data port
```

```
        IN      AL,DX                   ;Read misc register
        AND     AL,0F3h                 ;Clear addressing bits
        OR      AL,04h                  ;Enable A0000-AFFFF addressing
        OUT     DX,AL                   ;Output value

        ; Clean up and return to caller

        POP     ES                      ;Restore segment registers
        POP     DS
        POP     SI
        POP     DI

        MOV     SP,BP                   ;Standard C exit point
        POP     BP
        RET
_Select_Graphics ENDP

;***********************************************************************
;                                                                      *
; Select_Page                                                          *
; Entry:                                                               *
;       AL - Page number                                               *
;                                                                      *
;***********************************************************************

Select_Page     PROC NEAR
        CMP     AL,CS:RW_Page           ;Check if already selected
        JNE     SP_Go
        RET
SP_Go:
        PUSH    AX
        PUSH    DX

        AND     AL,7                    ;Force page number into range
        MOV     CS:RW_Page,AL           ;Save as most recent RW page
        MOV     CS:R_Page,0FFh          ;Invalidate R and W pages
        MOV     CS:W_Page,0FFh
        MOV     AH,AL                   ;Copy page number
        XOR     AH,02h                  ;Invert bit 1
        MOV     DX,EXTEND_REG_ADDR      ;Address of extended register bank
        MOV     AL,PAGE_REG             ;Index of select page register
        OUT     DX,AL                   ;Select the page register
        INC     DX                      ;Advance address to data
        IN      AL,DX                   ;Read previous value
        AND     AL,0F0h                 ;Preserve upper nibble
        OR      AL,AH                   ;Combine preserved bits with page number
        OUT     DX,AL                   ;Select new page

        POP     DX
        POP     AX
        RET
Select_Page     ENDP

;***********************************************************************
;                                                                      *
; Select_Read_Page                                                     *
; Entry:                                                               *
;       AL - Page number                                               *
;                                                                      *
;***********************************************************************

Select_Read_Page PROC NEAR
        CMP     AL,CS:R_Page            ;Check if already selected
        JNE     SRP_Go
        RET
SRP_Go:
        RET
Select_Read_Page ENDP
```

```
;*****************************************************************
;                                                                *
; Select_Write_Page                                              *
; Entry:                                                         *
;       AL - Page number                                         *
;                                                                *
;*****************************************************************

Select_Write_Page PROC NEAR
        CMP     AL,CS:W_Page            ;Check if already selected
        JNE     SWP_Go
        RET
SWP_Go:
        RET
Select_Write_Page ENDP

;*****************************************************************
;                                                                *
; Enable_Dual_Page                                               *
; Disable_Dual_Page                                              *
;                                                                *
; Entry:                                                         *
;      AL - Page number                                          *
;                                                                *
;*****************************************************************

Enable_Dual_Page PROC NEAR
        RET
Enable_Dual_Page ENDP

Disable_Dual_Page PROC NEAR
        RET
Disable_Dual_Page ENDP

;*****************************************************************
;                                                                *
; _Select_Page(PageNumber)                                       *
; Entry:                                                         *
;       PageNumber - Page number                                 *
;                                                                *
;*****************************************************************

Arg_PageNumber  EQU     BYTE PTR [BP+4]

_Select_Page    PROC NEAR
        PUSH    BP                      ;Setup frame pointer
        MOV     SP,BP
        MOV     AL,Arg_PageNumber       ;Fetch argument
        POP     BP                      ;Restore BP
        JMP     Select_Page
_Select_Page    ENDP

;*****************************************************************
;                                                                *
; _Select_Read_Page(PageNumber)                                  *
; Entry:                                                         *
;       PageNumber- Page number for read                         *
;                                                                *
;*****************************************************************

Arg_PageNumber  EQU     BYTE PTR [BP+4]

_Select_Read_Page       PROC NEAR
        PUSH    BP                      ;Setup frame pointer
        MOV     SP,BP
        MOV     AL,Arg_PageNumber       ;Fetch argument
        POP     BP                      ;Restore BP
        JMP     Select_Read_Page
_Select_Read_Page   ENDP
```

```
;*********************************************************************
;                                                                    *
; _Select_Write_Page(PageNumber)                                     *
; Entry:                                                             *
;       PageNumber - Page number for write                           *
;                                                                    *
;*********************************************************************

Arg_PageNumber  EQU     BYTE PTR [BP+4]

_Select_Write_Page      PROC NEAR
        PUSH    BP                      ;Setup frame pointer
        MOV     SP,BP
        MOV     AL,Arg_PageNumber       ;Fetch argument
        POP     BP                      ;Restore BP
        JMP     Select_Write_Page
_Select_Write_Page      ENDP

;*********************************************************************
;*                                                                   *
;* _Select_Text                                                      *
;*      Set VGA adapter to text mode                                 *
;*                                                                   *
;*********************************************************************

_Select_Text    PROC NEAR
        MOV     AX,TEXT_MODE            ;Select mode 3
        INT     10h                     ;Use BIOS to reset mode
        RET
_Select_Text    ENDP

Last_Byte:
_Text   ENDS
        END
```

# Detection and Identification

Everex recommends that their VGA boards be detected using the extended BIOS function call Return Emulation Status; a value of 70h should be returned in register AL. Code similar to that below can be used to detect Everex Viewpoint boards:

```
        MOV     AX,7000h        ;Function = 70, Sub-Function = 0
        MOV     BX,0            ;Extended fn = Get emulation status
        INT     10h             ;Call BIOS
        CMP     AL,70h          ;Check if AL set to proper code
        JNE     Not_Everex      ;...No, it is not Everex
        AND     DX,0FFF0h       ;Isolate board model number
        CMP     DX,06780h       ;Check for Everex Viewpoint
        JNE     Not_Veiwpoint   ;...No, not Viewpoint
Everex_Viewpoint_Found:
```

Note that this extended BIOS call will also return information about the type of monitor attached and the amount of display memory available. For more information about BIOS service 7000h see the "The BIOS," earlier in this chapter.

Trident recommends reading the Hardware Version register, and checking for a value of 1 or 2 in the lower nibble, to detect 8800BR and 8800CS chips. Code similar to that below can be used to read the Hardware Version register:

```
        MOV     DX,3C4h                 ;Address of extended reg bank
        MOV     AL,0Bh                  ;Index of version register
        OUT     DX,AL                   ;Select version register
        INC     DX
        IN      AL,DX                   ;Read version
        AND     AL,0Fh                  ;Keep only lower nibble
        CMP     AL,1                    ;Check for version 1
        JE      Trident_V1_Found
        CMP     AL,2                    ;Check for version 2
        JNE     Not_Trident
Trident_V2_Found:
```

# 17

# Tseng ET3000
# STB VGA EM-16

TSENG LABS INC

# Introduction

VGA Extra/EM and VGA Extra/EM-16 from STB are based on the ET3000 VGA chip made by Tseng Labs. Tseng Labs is a major supplier of VGA chips to board manufacturers, and the ET3000 can be found on boards from many different vendors.

VGA Extra is sold with either 256K or 512K of display memory. It can drive a VGA-compatible analog display, or it can drive a TTL color or monochrome display of the type used with MDA, CGA, or EGA adapters. This means a user can add a VGA adapter to his system without necessarily replacing his display. Not all VGA display modes can be supported with TTL displays, however. In addition the VGA compatibility, ET3000 also includes full hardware emulation for EGA, CGA, MDA and Hercules.

STB VGA EM/16 is supplemented with a full range of drivers for popular products such as MS-Windows, GEM and AutoCAD.

# New Display Modes

Table 17-1 lists the enhanced display modes that are supported by the VGA Extra EM. All modes listed can be selected using the BIOS mode select function.

**Table 17-1. Enhanced display modes—STB VGA Extra**

| Mode | Type | Resolution | Colors | Memory Required |
|---|---|---|---|---|
| 08h | Text | 132 col x 25 rows | mono | 256 KB |
| 0Ah | Text | 132 col x 44 rows | mono | 256 KB |
| 22h | Text | 132 col x 44 rows | 16 | 256 KB |
| 23h | Text | 132 col x 25 rows | 16 | 256 KB |
| 24h | Text | 132 col x 28 rows | 16 | 256 KB |
| 29h | Graphics | 800x600 | 16 | 256 KB |
| 2Dh | Graphics | 640x350 | 256 | 256 KB |
| 2Eh | Graphics | 640x480 | 256 | 512 KB |
| 30h | Graphics | 800x600 | 256 | 512 KB |
| 36h | Graphics | 960x720 | 16 | 512 KB |
| 37h | Graphics | 1024x768 | 16 | 512 KB |

Tseng Labs recommends that all manufacturers using the Tseng ET3000 chip support extended modes and mode numbers as listed in Table 17-2. Many boards based on Tseng VGA chips support these modes.

Table 17-2. Enhanced display modes—recommended by Tseng

| Mode | Type | Resolution | Colors | Memory Required | Display Type |
|---|---|---|---|---|---|
| 18h (1) | Text | 132 col x 44 rows | mono | 256 KB | SuperVGA |
| 19h (1) | Text | 132 col x 25 rows | mono | 256 KB | SuperVGA |
| 1Ah (1) | Text | 132 col x 28 rows | mono | 256 KB | SuperVGA |
| 22h | Text | 132 col x 44 rows | 16 | 256 KB | SuperVGA |
| 23h | Text | 132 col x 25 rows | 16 | 256 KB | SuperVGA |
| 24h | Text | 132 col x 28 rows | 16 | 256 KB | SuperVGA |
| 26h (1) | Text | 80 col x 60 rows | 16 | 256 KB | VGA |
| 25h | Graphics | 640x480 | 16 | 256 KB | VGA |
| 27h | Graphics | 720x512 | 16 | 256 KB | SuperVGA |
| 29h | Graphics | 800x600 | 16 | 256 KB | SuperVGA |
| 2Dh | Graphics | 640x350 | 256 | 512 KB | VGA |
| 2Eh | Graphics | 640x480 | 256 | 512 KB | VGA |
| 2Fh (1) | Graphics | 720x512 | 256 | 512 KB | SuperVGA |
| 30h | Graphics | 800x600 | 256 | 512 KB | SuperVGA |
| 37h | Graphics | 1024x768 | 16 | 512 KB | 8514 or XL |

Note: These modes are not documented on STB EM-16 (STB has additional modes 8, 0Ah and 36h).

# Memory Organization

For all extended display modes of the ET3000, display memory organization is closely patterned after standard IBM VGA display modes.

VGA Wonder includes a display memory paging mechanism that is needed in some display modes to make the entire display memory accessible to the processor. Display memory paging is described in detail later in this chapter.

## High Resolution Text Modes

These modes utilize memory maps that are similar to those used in standard text modes (modes 0,1,2,3 and 7), except that the number of characters per line, or number of lines per screen, is increased. Display memory is organized as shown in Figure 5-1 (see Chapter 5).

## 16-Color Graphics Modes

Memory organization for these modes resembles VGA mode 12h (640x480 16-color graphics), except that both the number of pixels per scan line and the number of scan

lines are increased. Display memory organization is shown in Figure 7-1. See Chapter 7 for programming examples.

For 1024x768 16-color graphics, display memory is mapped to the host as one 128K window at host memory address A000:0 to B000:FFFF. This configuration allows for efficient graphics programming, but limits the ability of the VGA Extra to co-reside with any other type of display adapter while this mode is being used. The number of memory pages that can be selected via the Segment Select register (I/O address 3CDh) is 64K

## 256-Color Graphics Modes

Memory organization for these modes resembles VGA mode 13h (320x200 256-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is shown in Figure 8-1. See Chapter 8 for programming examples.

# New Registers

Internal to the Tseng Labs ET3000 VGA chip is a bank of new registers that give the programmer access to added features in the device. These features include hardware zooming and display memory paging. The new registers are summarized in Table 17-3.

## Hardware Zoom Registers

"Zooming in" on the display magnifies a portion of the display screen. The reverse procedure, "zooming out", restores the full picture to the screen. These two functions are used extensively in applications such as desktop publishing and CAD/CAM, where the user must alternate between viewing an entire document or drawing and examining a portion of it in close detail.

With the VGA, display zooming is usually performed in software. VGA Extra includes hardware zooming which is very fast, but is too limited to satisfy the requirements of most applications. Many zooming functions must still be implemented in software.

Zooming is supported only in graphics modes, it is not fully supported in text modes. To learn more about zooming see "Hardware Zooming" later in this chapter.

Table 17-3. VGA EM-16 extended registers

| I/O Addr(Index) | Register | Function |
|---|---|---|
| 3C5 (6) | Zoom Control | Zoom Enable, Zoom factor |
| 3C5 (7) | TS Aux Mode | Select between EGA and VGA operation<br>Enable 8 simultaneous fonts |
| 3B5/3D5 (1Bh) | X Zoom Start | Sets X coordinate of zoom |
| 3B5/3D5 (1Ch) | X Zoom End | window |
| 3B5/3D5 (1Dh) | Y Zoom Start | Sets Y coordinate of zoom |
| 3B5/3D5 (1Eh) | Y Zoom End | window |
| 3B5/3D5 (1Fh) | Y Start/End High | |
| 3B5/3D5 (20h) | Zoom Start Low | Sets start address of |
| 3B5/3D5 (21h) | Zoom Start High | data in zoom window |
| 3B5/3D5 (23h) | Extended Start Address | |
| 3B5/3D5 (24h) | Compatibility Control | Enable CGA/MDA emulation |
| 3B5/3D5 (25h) | Overflow High | |
| 3C0 (16h) | Miscellaneous | Enable high resolution modes |
| 3CD | Page Select | Select page, and page size |
| 3DE | AT&T Mode Control | |

## Zoom Control Register (I/O Address 3C5, Index 6)

D7 - Zoom Enable (1 = zoom enabled)
D6-D4 - X Zoom Factor
D3 - Reserved
D2-D0 - Y Zoom Factor

X Zoom Factor sets the amount of magnification in the X direction.
Y Zoom Factor sets the amount of magnification in the Y direction.
Zoom Enable turns zooming on and off.

## X Zoom Start and End Registers (I/O Addr 3B5/3D5, Index 1Bh and 1Ch)

These registers define the corners of the on screen zoom window in the X direction (in character clock units), where the zoomed area will be displayed. X coordinates are selectable in increments of one character clock, which in all graphics modes corresponds to 8 pixels. These registers, which are 8 bits wide, are therefore sufficient to position the window anywhere on the screen. The value (Zoom End - Zoom_Start)/

(Zoom_Factor + 1) must be a positive integer. To display N characters zoomed 2x, the following formula can be used:

Zoom_End = Zoom_Start + (N - 1) * 2

### Y Zoom Start and End Registers (I/O Addr 3B5/3D5, Index 1Dh and 1Eh)

These registers define the corners of the on-screen zoom window in the Y direction, where the zoomed area will be displayed. Y coordinates, which are defined by scan line numbers, require more than eight bits each to define; a third register (Y Zoom Start and End Register High) is required. The value (Zoom End - Zoom_Start + 1)/(Zoom_Factor + 1) must be a positive integer. To display N characters zoomed 2x, the following formula can be used:

Zoom_End = Zoom_Start + N * Character_Height * 2 - 1

### Y Zoom Start & End Register High (I/O Addr 3B5/3D5 Index 1Fh)

D7-D4 - Y Zoom End High
D3-D0 - Y Zoom Start High

This register, in conjunction with the Y Zoom Start and Y Zoom End registers, defines the corners of the on-screen zoom window in the Y direction.

### Zoom Start Address Low and Mid (I/O Addr 3B5/3D5 Index 20h and 21h)

These registers define the starting address, offset from the zoom window, of data to be displayed in the zoom window. Setting the Start Address to zero, for example, will cause the upper left corner of the displayed zoomed window to coincide with the upper left corner of the character being zoomed.

Note that changing the linear Start Address registers of the CRT Controller (index 0Ch, 0Dh, or 23h) can force adjustments to the values used for the Zoom Start Address.

## Start Address Overflow Register (I/O Addr 3B5/3D5, Index 23h)

D7-D3 - Reserved
D2 - Bit 16 (MSB) of Zoom Address
D1 - Bit 16 (MSB) of Start Address
D0 - Bit 16 (MSB) of Cursor Address

In order to support the highest resolution display modes of the VGA Extra, an extra bit must be added to these VGA registers.

## Compatibility Control Register (I/O Address 3B5/3D5, Index 24h)

D7 - CGA/MDA/Hercules (enable 6845)
D6 - Enable Double Scan and Underline Attribute
D5 - Enable External ROM CRTC Address Translation
D4 - Reserved
D3 - Enable Input to A8 of 1MB DRAMs
D2 - Enable Tristate For all Output Pins
D1 - Additional Master Clock Select
D0 - Enable Clock Translate

## Auxiliary Overflow Register (I/O Address 3B5/3D5, Index 25h)

D7 - Enable Interlace (1 = enabled)
D6 - Reserved
D5 - Reserved
D4 - Line Compare Register Bit 10
D3 - Vertical Sync Start Register Bit 10
D2 - Vertical Display End Register Bit 10
D1 - Vertical Total Bit 10
D0 - Vertical Blank Start Bit 10

In order to support the highest resolution display modes of the VGA Extra, an extra bit must be added to these VGA registers.

Enable Interlace causes the VGA Extra to generate timing for interlaced displays. This is normally used to support 1024x768 resolution on 8514A-compatible interlaced displays.

## Segment Select Register (I/O Address 3CDh)

D7,D6 - Segment Configuration
D5-D3 - Read Segment Select
D2-D0 - Write Segment Select

SuperVGAs that include an increased amount of display memory require a memory paging mechanism in order to access all of display memory. The VGA Extra supports

two windows into display memory; one read only and one write only. Operation of these windows is controlled by the Segment Select register.

**Segment Configuration** defines the configuration of the memory paging logic. These two bits can be thought of as defining page size. By default, the value 00 is used for 16-color modes, defining a page size of 128K. A value of 01 is used in 256-color modes, defining a page size of 64K. In all programming examples in this book, page size is always assumed to be 64K; the value of bits 7 and 6 should always be set to 01. Table 17-4 shows the memory configurations that are supported.

Table 17-4. VGA Extra Memory Paging Configurations

| D7 | D6 | Memory Configuration |
|---|---|---|
| 0 | 0 | 8 segments of 128K each |
| 0 | 1 | 2 banks of 8 64K segments |
| 1 | 0 | 1 Megabyte of linear memory (no segments) |
| 1 | 1 | Invalid |

**Read Segment Select** selects which segment of display memory will be read from during CPU read operations. **Write Segment Select** selects which segment of display memory will be written to during CPU write operations. To learn more about the paging registers see the section titled "Display Memory Paging" later in this chapter.

## TS Auxiliary Mode (I/O Address 3C4, Index 7)

D7 - VGA Enable
D6 - MCLK/2
D5 - BIOS ROM Address Map 2
D4 - Enable Multiple Soft Font
D3 - BIOS ROM Address Map 1
D2 - Complement Split Screen Mode
D1 - Complement Zoom Window Mode
D0 - Complement Normal Window Mode

**Enable Multiple Soft Font** is used to enable eight simultaneous fonts in text modes. When this bit is set, bits D3, D4 and D6 in each character attribute byte determine which of the eight character generators should be used for that character (bit D5 must be set to the complement of D6). This register is normally locked (as are CRTC registers 0-7). To unlock this register, bit D7 of CRTC register 11h must be 0. To learn more about multiple fonts see Listing 17-3 in the programming examples at the end of this chapter.

## CRTC Vertical Sync End (Address 3D4h, Index 11h)

D7 - Protection bit
D6 - Reserved
D5 - Enable vertical interrupt when low
D4 - Clear vertical interrupt when low
D0 to D3 - Scan line at which vertical sync ends mod 16

**Protection bit** is used to enable access to CRTC registers 0 to 7, and to the TS Auxiliary Mode register. This bit must be set to zero enable access to the **Enable Multiple Soft Font** bit of the TS Aux register.

# Programming Examples

## Display Memory Paging

The display memory paging mechanism of the ET3000 maps selected portions of the display memory to the processor. Operation of display memory paging is very similar to the paging mechanism used for expanded memory boards (also called EMS or LIM memory). A 64K or 128K logical page of VGA RAM (a chunk of display memory) is mapped into the PC host address space in the normal VGA display memory address space. An I/O register (the Segment Select register at I/O address 3CDh), is used to define which pages of display memory are selected. In graphics modes, boards based on Tseng 3000 chips have two 64K pages mapped at A000:0, one for reading and one for writing. To learn more about dual pages see Chapter 5. The paging register for ET3000 based boards is illustrated in Figure 17-1.

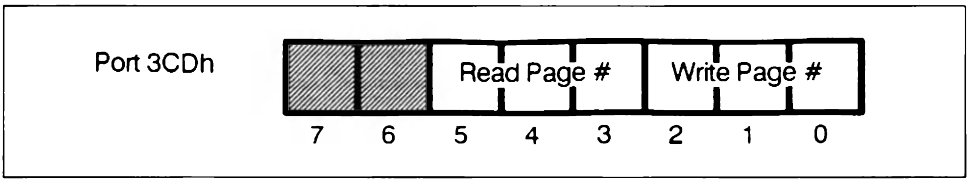


**Figure 17-1. Page Select register**

Listing 17-1 contains examples showing how to select paging registers on the VGA EM-16 board. Note that the paging routines in Listing 17-1 force a page size of 64K by setting bit D6 of the Segment Select register to 1. This is needed for proper operation of our 16-color drawing routines.

## Listing 17-1. File: TSENG\SELECT.ASM

```
;*********************************************************************
;* File:  SELECT.ASM                                                 *
;* Description: This module contains procedures to select mode and to *
;*              select pages.  It also initializes global variables   *
;*              according to the values in the MODE.INC include file. *
;* Entry Points:                                                      *
;*              _Select_Graphics   - Select a graphics mode           *
;*              _Select_Text       - Set VGA adapter into text mode   *
;*              _Select_Page       - Select read and write page       *
;*              _Select_Read_Page  - Select read page only            *
;*              _Select_Write_Page - Select write page only           *
;* Uses:                                                              *
;*              MODE.INC           - Mode dependent constants         *
;*              Following are modes and paths for TSENG 3000 boards:  *
;*     |----- 256 colors ------|  |-- 16 colors --|  4 colors  2 colors *
;*    640x400  640x480  800x600  800x600  1024x768  1024x768  1024x768 *
;* Mode:  N/A     2Eh      30h      29       37h       N/A       N/A   *
;* Path:  N/A   256COL   256COL    16COL     16COL      N/A       N/A   *
;*********************************************************************

     INCLUDE VGA.INC
     INCLUDE MODE.INC     ;Mode dependent constants

     PUBLIC    _Select_Graphics
     PUBLIC    _Select_Text
     PUBLIC    _Select_Page
     PUBLIC    _Select_Read_Page
     PUBLIC    _Select_Write_Page

     PUBLIC    Select_Page
     PUBLIC    Select_Read_Page
     PUBLIC    Select_Write_Page
     PUBLIC    Enable_Dual_Page
     PUBLIC    Disable_Dual_Page

     PUBLIC    Graf_Seg
     PUBLIC    Video_Height
     PUBLIC    Video_Width
     PUBLIC    Video_Pitch
     PUBLIC    Video_Pages
     PUBLIC    Ras_Buffer
     PUBLIC    Two_Pages

     PUBLIC    Last_Byte

;---------------------------------------------------------------------
; Data segment variables
;---------------------------------------------------------------------

;_DATA     SEGMENT WORD PUBLIC 'DATA'
;_DATA     ENDS

;---------------------------------------------------------------------
; Constant definitions
;---------------------------------------------------------------------

PAGE_SEL_PORT  EQU  3CDh            ;IO Address for page select register

;---------------------------------------------------------------------
; Code segment variables
;---------------------------------------------------------------------

_TEXT     SEGMENT BYTE PUBLIC 'CODE'

Graf_Seg       DW   0A000h          ;Graphics segment addresses
               DW   0A000h
OffScreen_Seg  DW   0A000h          ;First byte beyond visible screen
```

```
Video_Pitch     DW   SCREEN_PITCH      ;Number of bytes in one raster
Video_Height    DW   SCREEN_HEIGHT     ;Number of rasters
Video_Width     DW   SCREEN_WIDTH      ;Number of pixels in a raster
Video_Pages     DW   SCREEN_PAGES      ;Number of pages in the screen
Ras_Buffer      DB   1024 DUP (0)      ;Working buffer
R_Page          DB   0FFh              ;Most recently selected page
W_Page          DB   0FFh
RW_Page         DB   0FFh
Two_Pages       DB   CAN_DO_RW         ;Indicate separate R & W capability

;*********************************************************************
;*                                                                   *
;* _Select_Graphics(HorizPtr, VertPtr, ColorsPtr)                    *
;*      Initialize VGA adapter to 640x400 mode with                  *
;*      256 colors.                                                  *
;*                                                                   *
;* Entry:                                                            *
;*      None                                                         *
;*                                                                   *
;* Returns:                                                          *
;*      VertPtr   - Vertical resolution                              *
;*      HorizPtr  - Horizontal resolution                            *
;*      ColorsPtr - Number of supported colors                       *
;*                                                                   *
;*********************************************************************

Arg_HorizPtr   EQU  WORD PTR [BP+4] ;Formal parameters
Arg_VertPtr    EQU  WORD PTR [BP+6] ;Formal parameters
Arg_ColorsPtr  EQU  WORD PTR [BP+8] ;Formal parameters

_Select_Graphics PROC NEAR
        PUSH    BP                          ;Standard C entry point
        MOV     BP,SP

        PUSH    DI                          ;Preserve segment registers
        PUSH    SI
        PUSH    DS
        PUSH    ES

        ; Select graphics mode

        MOV     AX,GRAPHICS_MODE            ;Select graphics mode
        INT     10h

        ; Reset 'last selected page'

        MOV     AL,0FFh                     ;Use 'non-existent' page number
        MOV     CS:R_Page,AL                ;Set currently selected page
        MOV     CS:W_Page,AL
        MOV     CS:RW_Page,AL

        ; Set return parameters

        MOV     SI,Arg_VertPtr              ;Fetch pointer to vertical resolution
        MOV     WORD PTR [SI],SCREEN_HEIGHT ;Set vertical resolution
        MOV     SI,Arg_HorizPtr             ;Fetch pointer to horizontal resolution
        MOV     WORD PTR [SI],SCREEN_WIDTH  ;Set horizontal resolution
        MOV     SI,Arg_ColorsPtr            ;Fetch pointer to number of colors
        MOV     WORD PTR [SI],SCREEN_COLORS ;Set number of colors

        ; Clean up and return to caller

        POP     ES                         ;Restore segment registers
        POP     DS
        POP     SI
        POP     DI

        MOV     SP,BP                      ;Standard C exit point
        POP     BP
        RET
_Select_Graphics ENDP
```

```
;*********************************************************************
;                                                                    *
; Select_Page                                                        *
; Entry:                                                             *
;      AL - Page number                                              *
;                                                                    *
;*********************************************************************

Select_Page     PROC NEAR
       CMP      AL,CS:RW_Page               ;Check if already selected
       JNE      SP_Go
       RET
SP_Go:
       PUSH     AX
       PUSH     DX
       MOV      DX,PAGE_SEL_PORT            ;Fetch address of page select
       AND      AL,7                        ;Force page number into 0-7
       MOV      CS:RW_Page,AL               ;Save most recently selected page
       MOV      CS:R_Page,0FFh
       MOV      CS:W_Page,0FFh
       MOV      AH,AL                       ;Copy page into AH
       SHL      AH,1                        ;Shift page number
       SHL      AH,1
       SHL      AH,1
       OR       AL,AH                       ;Move page number into ""write" bits
       OR       AL,40h                      ;Force bit 6
       OUT      DX,AL                       ;Write out the new page select
       POP      DX
       POP      AX
       RET
Select_Page     ENDP

;*********************************************************************
;                                                                    *
; Select_Read_Page                                                   *
; Entry:                                                             *
;      AL - Page number                                              *
;                                                                    *
;*********************************************************************

Select_Read_Page PROC NEAR
       CMP      AL,CS:R_Page                ;Check if already selected
       JNE      SRP_Go
       RET
SRP_Go:
       PUSH   AX
       PUSH   DX
       AND    AL,7                          ;Force page number into 0-7
       MOV    AH,AL                         ;Copy page number into AH
       MOV    CS:R_Page,AH                  ;Save most recently selected page
       SHL    AH,1                          ;Shift page number
       SHL    AH,1
       SHL    AH,1
       MOV    DX,PAGE_SEL_PORT              ;Fetch address of page select
       IN     AL,DX                         ;Get current values
       AND    AL,07h                        ;Preserve bits 0-2
       OR     AL,40h                        ;Force bits 6 and 7
       OR     AL,AH                         ;Move page number into ""write'' bits
       OUT    DX,AL                         ;Write out the new page select
       MOV    CS:RW_Page,0FFh
       ; Clean up and return
       POP    DX
       POP    AX
       RET
Select_Read_Page ENDP
```

```
;***********************************************************************
;                                                                      *
; Select_Write_Page                                                    *
; Entry:                                                               *
;       AL - Page number                                               *
;                                                                      *
;***********************************************************************

Select_Write_Page PROC NEAR
       CMP     AL,CS:W_Page              ;Check if already selected
       JNE     SWP_Go
       RET
SWP_Go:
       PUSH    AX                        ;Preserve page number (AX gets trashed)
       PUSH    DX
       AND     AL,7                      ;Force page number into 0-7
       MOV     AH,AL                     ;Copy page number into AH
       MOV     CS:W_Page,AH              ;Save most recently selected page
       MOV     DX,PAGE_SEL_PORT          ;Fetch address of page select
       IN      AL,DX                     ;Get current values
       AND     AL,38h                    ;Preserve bits 3-5
       OR      AL,40h                    ;Force bits 6 & 7
       OR      AL,AH                     ;Move page number into ""read'' bits
       OUT     DX,AL                     ;Write out the new page select
       MOV     CS:RW_Page,0FFh
       ; Clean up and return
       POP     DX
       POP     AX
       RET
Select_Write_Page ENDP

;***********************************************************************
;                                                                      *
; _Select_Page(PageNumber)                                             *
; Entry:                                                               *
;       PageNumber - Page number                                       *
;                                                                      *
;***********************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Page   PROC NEAR
       PUSH    BP                        ;Setup frame pointer
       MOV     SP,BP
       MOV     AL,Arg_PageNumber         ;Fetch argument
       POP     BP                        ;Restore BP
       JMP     Select_Page
_Select_Page   ENDP

;***********************************************************************
;                                                                      *
; _Select_Read_Page(PageNumber)                                        *
; Entry:                                                               *
;       PageNumber- Page number for read                               *
;                                                                      *
;***********************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Read_Page   PROC NEAR
       PUSH    BP                        ;Setup frame pointer
       MOV     SP,BP
       MOV     AL,Arg_PageNumber         ;Fetch argument
       POP     BP                        ;Restore BP
       JMP     Select_Read_Page
_Select_Read_Page  ENDP
```

```
;*************************************************************************
;                                                                        *
; _Select_Write_Page(PageNumber)                                         *
; Entry:                                                                 *
;       PageNumber - Page number for write                               *
;                                                                        *
;*************************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Write_Page  PROC NEAR
       PUSH   BP                            ;Setup frame pointer
       MOV    SP,BP
       MOV    AL,Arg_PageNumber             ;Fetch argument
       POP    BP                            ;Restore BP
       JMP    Select_Write_Page
_Select_Write_Page  ENDP

;*************************************************************************
;*                                                                       *
;* _Select_Text                                                          *
;*      Set VGA adapter to text mode                                     *
;*                                                                       *
;*************************************************************************

_Select_Text   PROC NEAR
       MOV     AX,TEXT_MODE                 ;Select mode 3
       INT     10h                          ;Use BIOS to reset mode
       RET
_Select_Text   ENDP

;*************************************************************************
;*                                                                       *
;* Enable_Dual_Page                                                      *
;* Disable_Dual_Page                                                     *
;*      Not supported by Tseng based boards                              *
;*                                                                       *
;*************************************************************************

Enable_Dual_Page    PROC NEAR
     RET
Enable_Dual_Page    ENDP

Disable_Dual_Page   PROC NEAR
     RET
Disable_Dual_Page   ENDP

Last_Byte:
_Text     ENDS
     END
```

## Hardware Zooming

ET3000 provides the capability to zoom, in graphics modes, any block on the screen with dimensions 8xN by M. This block is magnified by a specified factor, and displayed at a specified position on the screen, by setting the appropriate extended registers as illustrated in Figure 17-2. To learn more about these registers see "Hardware Zoom Registers" earlier in this chapter.

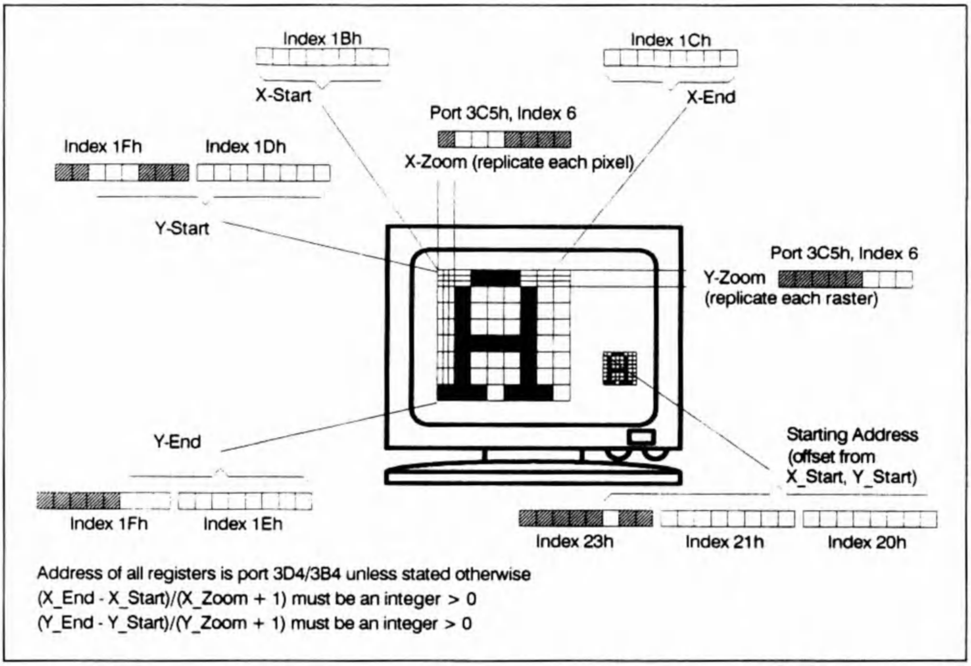


**Figure 17-2. Zoom registers**

Listing 17-2 demonstrates the hardware zoom feature of the VGA EM-16. It can be modified to operate in any graphics mode by changing the mode select number at the beginning of the program. While the program is running, arrow keys on the keypad can be used to change the origin of the magnified area. An area two characters wide and two characters high, with the upper left corner at the origin, is magnified and displayed at the same origin. The magnification factor can be increased with the " + " key on the keypad, and decreased with the "-" key on the keypad.

## Listing 17-2. File: TSENG\ZOOM.ASM

```
;**********************************************************************
;*                                                                    *
;* File:        ZOOM.ASM                                              *
;* Description: This module contains a program to demonstrate a use   *
;*              of enhanced zoom capabilities of the board. Routines  *
;*              are provided to define and move around 8x8 zoom window. *
;*              Cursor keys to move window, <+> and <-> keypag keys to *
;*              set zoom factor and <ESC> key to restore zoom and quit. *
;*                                                                    *
;* Entry Points:                                                      *
;*                                                                    *
;* Uses:                                                              *
;*                                                                    *
;**********************************************************************

     INCLUDE VGA.INC

;**********************************************************************
; Scan code definition
;**********************************************************************

ESC_KEY         EQU  01h
LEFT_KEY        EQU  4Bh
RIGHT_KEY       EQU  4Dh
UP_KEY          EQU  48h
DOWN_KEY        EQU  50h
PLUS_KEY        EQU  4Eh
MINUS_KEY       EQU  4Ah

CRTC_PORT       EQU  3D4h
ZOOM_CONTROL    EQU  06h
ZOOM_X_REG      EQU  1Bh
ZOOM_Y_REG      EQU  1Dh
ZOOM_LOW_REG    EQU  20h

WINDOW_WIDTH    EQU  2                  ;Zoom window size in characters
WINDOW_HEIGHT   EQU  2

;**********************************************************************
; Main program
;**********************************************************************

_TEXT    SEGMENT BYTE PUBLIC 'CODE'
         ASSUME  CS:_TEXT, ES:NOTHING, DS:_TEXT, SS:_STACK
Zoom     PROC    FAR
         PUSH    DS                     ;Save return address
         XOR     AX,AX
         PUSH    AX

         MOV     AX,CS
         MOV     DS,AX                  ;Set Data seg to Code seg

         ; Force into graphics mode 12h (any graphics mode will do
         ; since rest of the progrm should behave properly for any mode)

         MOV     AX,12h                 ;Set for mode 12h, function 0
         INT     10h                    ;Use BIOS for force mode 3

         MOV     DX,0E20h               ;Fill screen with data
         MOV     BH,0
         MOV     BL,7
         MOV     CX,2000
Fill_Loop:
         MOV     AX,DX
         INT     10h
         INC     DL
         JNZ     Skip
         MOV     DL,20h
```

```
Skip:
        LOOP    Fill_Loop

        ; Compute dimensions of the screen

        XOR     AX,AX                   ;Point segment to BIOS data area
        MOV     ES,AX
        MOV     SI,044Ah                ;Offset of COLUMNS parameter
        MOV     AX,ES:[SI]              ;Fetch number of text columns
        MOV     Screen_Width,AX         ;Save for later
        MOV     SI,0484h                ;Offset of ROWS parameter
        MOV     AL,ES:[SI]              ;Fetch number of rows-1
        INC     AL
        MOV     Screen_Rows,AX          ;Save for later
        MOV     SI,0485h                ;Number of scanlines per character
        MOV     AX,ES:[SI]
        MOV     Char_Height,AX

        MUL     Screen_Rows             ;Compute number of scanlines
        MOV     Screen_Height,AX

        ; Set up initial zoom (factor=3, position 0, source 0,0)

        CALL    Set_Factor              ;Set the zoom factor
        CALL    Set_Window              ;Set window size and position

        ; Get next key and jump according to key pressed

Get_Next_Key:
        MOV     AH,0                    ;Function = read next character
        INT     16h                     ;Use BIOS to get next key

        CMP     AH,ESC_KEY              ;Check for escape key
        JNE     Not_Esc
        JMP     Zoom_Disable
Not_Esc:
        CMP     AH,LEFT_KEY             ;Check for left arrow
        JNE     Not_Left
        DEC     Zoom_X                  ;Update x position
        CALL    Set_Window              ;Set window size and position
        JMP     Get_Next_Key
Not_Left:
        CMP     AH,RIGHT_KEY            ;Check for rigth arrow
        JNE     Not_Right
        INC     Zoom_X                  ;Update y position
        CALL    Set_Window              ;Set window size and position
        JMP     Get_Next_Key
Not_Right:
        CMP     AH,UP_KEY               ;Check for up arrow
        JNE     Not_Up
        MOV     AX,Char_Height          ;Update y position
        SUB     Zoom_Y,AX
        CALL    Set_Window              ;Set window size and position
        JMP     Get_Next_Key
Not_Up:
        CMP     AH,DOWN_KEY             ;Check for down arrow
        JNE     Not_Down
        MOV     AX,Char_Height          ;Update y position
        ADD     Zoom_Y,AX
        CALL    Set_Window              ;Set window size and position
        JMP     Get_Next_Key
Not_Down:
        CMP     AH,PLUS_KEY             ;Check for keypad plus
        JNE     Not_Plus
        INC     Zoom_Factor             ;Update zoom factor
        CALL    Set_Factor              ;Set new zoom factor
        JMP     Get_Next_Key
Not_Plus:
        CMP     AH,MINUS_KEY            ;Check for keypad minus
        JNE     Not_Minus
        DEC     Zoom_Factor             ;Update zoom factor
```

```
         CALL   Set_Factor              ;Set new zoom factor
         JMP    Get_Next_Key
Not_Minus:
         JMP    Get_Next_Key

         ; Disable zoom

Zoom_Disable:
         MOV    DX,SEQUENCER_PORT       ;Address of sequencer
         MOV    AX,ZOOM_CONTROL         ;Index of control register
         OUT    DX,AX                   ;Set zoom factor to 1 and disable

         ; Clean up and exit

Zoom_Done:
         RET                            ;Exit
Zoom     ENDP

;************************************************************************
;                                                                       *
; Set_Factor                                                            *
;        Set the x and y zoom factor and enable zoom                    *
;                                                                       *
; Entry: DS:Zoom_Factor - Zoom factor                                   *
;                                                                       *
;************************************************************************

Set_Factor      PROC NEAR
       ; Force factor into range 0-7

       MOV     AX,Zoom_Factor           ;Fetch factor
       CWD                              ;Check if factor negative (DX=FFFF)
       NOT     DX                       ;(DX=0000 if factor was negative)
       AND     AX,DX                    ;and set factor to zero if negative
       SUB     AX,7                     ;Check if factor greater than 7
       CWD                              ;(DX=0000 if factor >= 7)
       AND     AX,DX                    ;(AX=0000 if factor >= 7)
       ADD     AX,7                     ;and set factor to 7 if over 7
       MOV     Zoom_Factor,AX           ;Save (adjusted factor)

       ; Select new factor

       MOV     AH,AL                    ;Copy factor into bits 4-6
       SHL     AH,1
       SHL     AH,1
       SHL     AH,1
       SHL     AH,1
       OR      AH,AL                    ;Combine x and y factors
       OR      AH,80h                   ;Combine with 'enable' bit
       MOV     DX,SEQUENCER_PORT        ;Address of sequencer
       MOV     AL,ZOOM_CONTROL          ;Index of control register
       OUT     DX,AX                    ;Select zoom factor and enable

       ; Change window size

       CALL    Set_Window               ;Change window size and position

       RET
Set_Factor     ENDP

;************************************************************************
;                                                                       *
; Set_Window                                                            *
;      Set the x and y position of the displayed window                 *
;                                                                       *
; Entry: DS:Zoom_X  - Window position                                   *
;        DS:Zoom_Y                                                      *
;                                                                       *
;************************************************************************

Set_Window      PROC NEAR
```

```
; Force x position into range 0 to maxx,
; where maxx = Screen_Width - WINDOW_WIDTH * (Zoom_Factor+1) - 1

MOV     AX,Zoom_Factor          ;Use zoom factor and width to
INC     AX                      ;    compute maxx
MOV     BX,WINDOW_WIDTH
MUL     BX
NEG     AX
ADD     AX,Screen_Width
DEC     AX
MOV     CX,AX                   ;Keep maxx in CX

MOV     AX,Zoom_X               ;Fetch x position
CWD                             ;Check if negative (DX=FFFF)
NOT     DX                      ;(DX=0000 if negative)
AND     AX,DX                   ;and set to zero if negative
SUB     AX,CX                   ;Check if greater than max
CWD                             ;(DX=0000 if >= max)
AND     AX,DX                   ;(AX=0000 if >= max)
ADD     AX,CX                   ;and set to max if over max
MOV     Zoom_X,AX               ;Save (adjusted position)

; Force y position into range 0 to maxy,
; where maxy = Screen_Height - WINDOW_HEIGHT * (ZoomFactor+1) - 1

MOV     AX,Zoom_Factor          ;Use zoom factor and width to
INC     AX                      ;    compute max
MOV     BX,WINDOW_HEIGHT
MUL     BX
MUL     Char_Height
NEG     AX
ADD     AX,Screen_Height
DEC     AX
MOV     CX,AX                   ;Keep max in CX

MOV     AX,Zoom_Y               ;Fetch y position
CWD                             ;Check if negative (DX=FFFF)
NOT     DX                      ;(DX=0000 if negative)
AND     AX,DX                   ;and set to zero if negative
SUB     AX,CX                   ;Check if greater than max
CWD                             ;(DX=0000 if >= max)
AND     AX,DX                   ;(AX=0000 if >= max)
ADD     AX,CX                   ;and set to max if over max
MOV     Zoom_Y,AX               ;Save (adjusted position)

Set new window x-start and x-end

MOV     BX,Zoom_X               ;Fetch x postion
MOV     DX,CRTC_PORT            ;Address of CRTC
MOV     AL,ZOOM_X_REG           ;Index of x-start register
MOV     AH,BL
OUT     DX,AX                   ;Select new x start

MOV     AX,Zoom_Factor          ;Compute window width
INC     AX
MOV     BX,WINDOW_WIDTH-1
MUL     BX
ADD     AX,Zoom_X               ;Compute window end
MOV     AH,AL
MOV     DX,CRTC_PORT            ;Address of CRTC
MOV     AL,ZOOM_X_REG+1         ;Index of x-end register
OUT     DX,AX                   ;Select new x-end

; Set new window y-start and y-end

MOV     BX,Zoom_Y               ;Fetch y position
SHL     BH,1                    ;Move high order bits into bits 3-5
SHL     BH,1
SHL     BH,1
MOV     DX,CRTC_PORT            ;Address of CRTC
MOV     AL,ZOOM_Y_REG           ;Index of y-start register
```

```
        MOV     AH,BL
        OUT     DX,AX                   ;Select new y start low
        ADD     AL,2                    ;Index of y start hi
        OUT     DX,AL                   ;Read previous value
        INC     DX
        IN      AL,DX
        AND     AL,NOT 38h              ;Clear previous value
        OR      AL,BH                   ;Move in new value
        OUT     DX,AL                   ;Set the new y start high

        MOV     AX,Zoom_Factor          ;Compute window height
        INC     AX
        MOV     BX,WINDOW_WIDTH
        MUL     BX
        MUL     Char_Height
        ADD     AX,Zoom_Y               ;Compute window end
        DEC     AX
        MOV     BL,AH                   ;Save y start high
        MOV     AH,AL                   ;Save y start low
        MOV     DX,CRTC_PORT            ;Address of CRTC
        MOV     AL,ZOOM_Y_REG+1         ;Index of y-end register
        OUT     DX,AX                   ;Select new y-end low
        INC     AL                      ;Index of y start hi
        OUT     DX,AL                   ;Read previous value
        INC     DX
        IN      AL,DX
        AND     AL,NOT 07h              ;Clear previous value
        OR      AL,BL                   ;Move in new value
        OUT     DX,AL                   ;Set the new y start high

        ; Set new zoom address (always set to zero, to force zoom source
        ; upper-left corner to be same upper-left corner of displayed window)

        MOV     DX,CRTC_PORT            ;Address of CRTC
        MOV     AL,ZOOM_LOW_REG         ;Index of start low
        MOV     AH,0                    ;Value
        OUT     DX,AX
        INC     AL                      ;Point to start mid
        OUT     DX,AX
        ADD     AL,2                    ;Point to start hi
        OUT     DX,AL                   ;Select hi
        INC     DX
        IN      AL,DX                   ;Read previous value
        AND     AL,NOT 04h              ;Clear previous value
        OUT     DX,AL                   ;Write new value

        ; Clean up and return

        RET
Set_Window      ENDP

;*************************************************************************
; Data definition                                                        *
;*************************************************************************

Zoom_Factor     DW    2                 ;Zoom factor - 1
Zoom_X          DW    0                 ;Position of displayed window
Zoom_Y          DW    0
Zoom_Address    DD    0                 ;Address of area to zoom

Scree_Width     DW    80                ;Number of columns on the screen
Screen_Height DW      400               ;Number of scanlines on the screen
Screen_Rows     DW    25                ;Number of text rows
Char_Height  DW    16                  ;Character height

_TEXT           ENDS

_STACK          SEGMENT PARA STACK 'STACK'
        DB      100h DUP(?)
_STACK          ENDS
        END
```

## Displaying Eight Simultaneous Fonts

Listing 17-3 demonstrates how to enable eight simultaneous fonts, how to download fonts, and how display text using all eight fonts. The program starts by creating seven new fonts from the standard 8x14 and 8x8 fonts, making normal, bold, italicized, and inverted fonts from each. Each font is copied into plane 2, using procedure Load_CG. This procedure enables plane 2 for writing, and disables the odd/even addressing used in text modes, before the font is written into memory. Fonts are loaded as indicated in Table 17-5. When multiple fonts are enabled, using TS Aux register at address 3C4h, index 06h, then each attribute byte determines color and font as is illustrated in Figure 17-3.

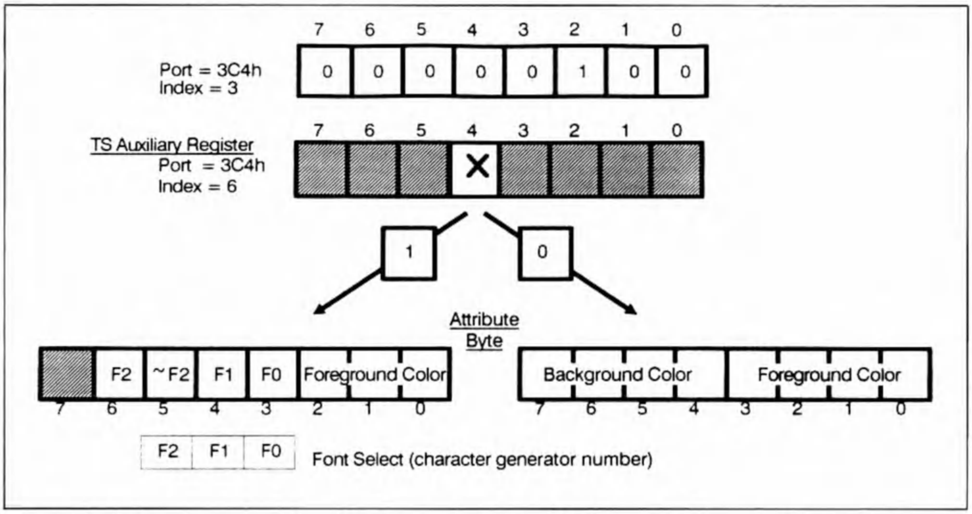


Figure 17-3. Multiple fonts

Table 17-5. Font locations in Plane 2

| Font Number | Offset in Plane 2 |
|---|---|
| 0 | 0 |
| 1 | 16K |
| 2 | 32K |
| 3 | 48K |
| 4 | 8K |
| 5 | 24K |
| 6 | 40K |
| 7 | 56K |

Show_Text is used to display text in each font. For each font a label is displayed, followed by 26 upper-and lower case characters, and numbers. Each font is displayed using BIOS service 13h, Write Text String, with the attribute set for the specified font.

**Listing 17-3. File: TSENG\TEXT.ASM**

```
;**************************************************************************
;*                                                                        *
;* File:        TEXT.ASM - Load 8 simultaneous fonts                      *
;* Description: A program to load 8 character generators and to display *
;*              eight simultaneous fonts.                                 *
;*              It is assumed that color VGA monitor is attached to VGA.*
;*                                                                        *
;**************************************************************************

      INCLUDE VGA.INC

_TEXT SEGMENT BYTE PUBLIC 'CODE'
      ASSUME  CS:_TEXT, ES:NOTHING, DS:NOTHING, SS:_STACK

Text  PROC    FAR
      PUSH    DS                         ;Save return address
      XOR     AX,AX
      PUSH    AX

      MOV     AX,CS
      MOV     DS,AX                      ;Set Data seg to Code seg

      ; Force into text mode 3

      MOV     AX,3                       ;Set for mode 3, function 0
      INT     10h                        ;Use BIOS for force mode 3

      ; Fetch 8x14 character generator and load it bold as char gen 1

      MOV     AX,1130h                   ;Fn=Char Gen, SubFn=Get Info
      MOV     BH,2                       ;Get info on 8x14
      INT     10h                        ;Use BIOS to get pointer to CG
      MOV     DL,14                      ;Character height
      CALL    Make_Bold                  ;Convert char gen to bold
      MOV     DI,4000h                   ;Load at 16k
      CALL    Load_CG

      ; Fetch 8x14 character generator and load it italicized as char gen 2

      MOV     AX,1130h                   ;Fn=Char Gen, SubFn=Get Info
      MOV     BH,2                       ;Get info on 8x14
      INT     10h                        ;Use BIOS to get pointer to CG
      MOV     DL,14                      ;Character height
      CALL    Make_Italics               ;Italicize the char gen
      MOV     DI,8000h                   ;Load at 32k
      CALL    Load_CG

      ; Fetch 8x14 character generator and load it inverted as char gen 3

      MOV     AX,1130h                   ;Fn=Char Gen, SubFn=Get Info
      MOV     BH,2                       ;Get info on 8x14
      INT     10h                        ;Use BIOS to get pointer to CG
      MOV     DL,14                      ;Character height
      CALL    Make_Inverted              ;Invert the char gen
      MOV     DI,0C000h                  ;Load at 48k
      CALL    Load_CG

      ; Fetch 8x8 character generator and load it as char gen 4

      MOV     AX,1130h                   ;Fn=Char Gen, SubFn=Get Info
      MOV     BH,3                       ;Get info on 8x8
      INT     10h                        ;Use BIOS to get pointer to CG
```

```
MOV     DL,8
MOV     DI,2000h                 ;Load at 8k
CALL    Load_CG

; Fetch 8x8  character generator and load it bold as char gen 5

MOV     AX,1130h                 ;Fn=Char Gen, SubFn=Get Info
MOV     BH,3                     ;Get info on 8x8
INT     10h                      ;Use BIOS to get pointer to CG
MOV     DL,8                     ;Character height
CALL    Make_Bold                ;Convert char gen to bold
MOV     DI,06000h                ;Load at 24k
CALL    Load_CG

; Fetch 8x8 character generator and load it italicized as char gen 6

MOV     AX,1130h                 ;Fn=Char Gen, SubFn=Get Info
MOV     BH,3                     ;Get info on 8x8
INT     10h                      ;Use BIOS to get pointer to CG
MOV     DL,8                     ;Character height
CALL    Make_Italics             ;Italicize the char gen
MOV     DI,0A000h                ;Load at 40k
CALL    Load_CG

; Fetch 8x8 character generator and load it inverted as char gen 7

MOV     AX,1130h                 ;Fn=Char Gen, SubFn=Get Info
MOV     BH,3                     ;Get info on 8x8
INT     10h                      ;Use BIOS to get pointer to CG
MOV     DL,8                     ;Character height
CALL    Make_Inverted            ;Invert the char gen
MOV     DI,0E000h                ;Load at 56k
CALL    Load_CG

; Enable multiple character fonts

;Unlock CRTC and TS Aux registers
MOV     DX,3D4h                  ;Address of CRTC
MOV     AL,11h                   ;Index of 'unlock' register
OUT     DX,AL                    ;Select register
INC     DX
IN      AL,DX                    ;Read current value
AND     AL,NOT 80h               ;CLEAR protection bit
OUT     DX,AL                    ;Write new value - Enable Access

;Set 'Enable multiple font' bit in TS Aux
MOV     DX,3C4h                  ;Address of TS Auxilliary register
MOV     AL,7                     ;Index of TS Aux
OUT     DX,AL                    ;Select TS Aux
INC     DX
IN      AL,DX                    ;Read current value
OR      AL,10h                   ;Set font enable bit
OUT     DX,AL                    ;Enable multiple fonts
DEC     DX

;Set 'Character Generator Select' to 'A .NE. B'
MOV     AX,0403h                 ;Index=Char Gen Sel reg, Data=4
OUT     DX,AX                    ;Enable two char generators

;Disable plane 3 from being displayed
MOV     DX,3DAh                  ;Disable plane 3 from display it
IN      AL,DX                    ;    first reset flip/flop
MOV     DX,3C0h                  ;    get address of Attr Ctrl
MOV     AL,32h                   ;    index of Color Plane Enable
OUT     DX,AL
MOV     AL,7                          enable only 3 planes (0-2)
OUT     DX,AL

; Display title line

MOV     BX,0007h                 ;Page=0, attribute=07h (font=0, color=7)
```

```
        MOV     CX,20                       ;20 characters
        MOV     DX,011Eh                    ;Row=01, column=30
        LEA     BP,Title_Msg                ;Fetch pointer to string
        MOV     AX,CS
        MOV     ES,AX
        MOV     AX,1300h                    ;Fn=String, SubFn=Use BL for attr.
        INT     10h                         ;Display the string

        ; Loop over fonts, displaying message in seven colors for each font

        XOR     BX,BX                       ;Set counter of fonts to do
Font_Loop:
        PUSH    BX                          ;Preserve counter, & put font # on stack
        SHL     BX,1                        ;Convert counter to index
        PUSH    CS                          ;Put address of text on the stack
        PUSH    WORD PTR CS:MSG_Ptr[BX]
        CALL    Show_Text                   ;Draw next set of text
        ADD     SP,4                        ;'Pop' text address
        POP     BX                          ;Restore counter
        INC     BX                          ;Update index
        CMP     BX,8                        ;Check if all fonts done
        JL      Font_Loop                   ;Go do next font if needed

        ; Wait for a key to be pressed

        MOV     AH,00h                      ;Function return key
        INT     16h                         ;Use BIOS to get the key

        ; Disable multiple character fonts

        MOV     DX,3C4h                     ;Address of TS Auxilliary register
        MOV     AL,7                        ;Index of TS Aux
        OUT     DX,AL                       ;Select TS Aux
        INC     DX
        IN      AL,DX                       ;Read current value
        AND     AL,NOT 10h                  ;Clear font enable bit
        OUT     DX,AL                       ;Disable multiple fonts

        ; Clean up and exit

Sho_Done:
        RET                                 ;Exit
Text  ENDP

;************************************************************************
;                                                                       *
; Show Text (font, text)                                                *
;    Display 'text' as a label, followed by 62 characters of alphabet   *
;    in row '2*font' with color 'font+1'                                *
;                                                                       *
;************************************************************************

Arg_Text  EQU  DWORD PTR [BP+4]
Arg_Font  EQU  BYTE PTR [BP+8]

Font_To_Attr    DB   00100000b,00101000b,00110000b,00111000b
                DB   01000000b,01001000b,01010000b,01011000b

Alphabet  DB    'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ'
          DB    '0123456789'

Show_Text PROC NEAR
     PUSH BP
     MOV  BP,SP

     ; Convert font number to attribute value

     MOV  AL,Arg_Font                  ;Translate font to attr bit3 = bit0
     LEA  BX,Font_To_Attr              ;                       bit4 = bit1
     XLAT CS:Font_To_Attr              ;                       bit5 =~bit2
     MOV  BL,AL                        ;                       bit6 = bit2
```

```
     MOV  BH,Arg_Font               ;Use color 1-4
     AND  BH,3                      ;   (Font AND 3) + 1
     INC  BH
     ADD  BL,BH

     ; Setup parameters for BIOS service call and call it to show label

     MOV  BH,0                      ;Page 0
     MOV  CX,18                     ;16 characters
     MOV  DH,Arg_Font               ;Compute starting row
     SHL  DH,1                      ;    as 'font*2 + 3'
     ADD  DH,3
     MOV  DL,0                      ;Starting column
     LES  BP,Arg_Text               ;Fetch pointer to string
     MOV  AX,1300h                  ;Fn=String, SubFn=Use BL for attr.
     INT  10h                       ;Display the string

     ; Alphabet

     ADD  DL,18                     ;Start in column 18
     MOV  CX,62                     ;64 characters to show
     LEA  BP,CS:Alphabet            ;Pointer to alphabet string
     MOV  AX,1300h                  ;Fn=String, SubFn=Use BL for attr.
     INT  10h                       ;Display string

     ; Clean up and exit

     POP  BP
     RET
Show_Text ENDP

;*************************************************************************
;                                                                        *
; Make_Bold                                                              *
;    Convert 8xN character genertor to bold, by shifting each byte       *
;    to the right and ORing it with the original.                        *
;                                                                        *
; Entry: DL -   Number of bytes in each character                        *
;     ES:BP - Pointer to the character generator                         *
;                                                                        *
;*************************************************************************


New_CG    DB   16*256 DUP (0)         ;Buffer for new char gen

Make_Bold PROC NEAR

     ;Setup counters

     MOV  BX,256                    ;Set counter of characters
     XOR  CH,CH                     ;Set counter of bytes
     MOV  AX,ES                     ;Set pointer to source
     MOV  DS,AX
     MOV  SI,BP
     LEA  DI,CS:New_CG              ;Set pointer to destination
     MOV  AX,CS
     MOV  ES,AX

     ; Loop over characters to change

MB_Char_Loop:
     MOV  CL,DL                     ;Set counter of bytes

     ; Loop over bytes to change

MB_Byte_Loop:
     LODSB                          ;Fetch original byte
     MOV  AH,AL                     ;Get a copy of the byte
     SHR  AL,1                      ;Shift byte to the right
     OR   AL,AH                     ;Combine bytes to make bold char
     STOSB                          ;Save new character
```

```
     LOOP MB_Byte_Loop                ;Check if all bytes done
     DEC  BX
     JG   MB_Char_Loop                ;Check if all chars done

     ; Clean up and exit

     LEA  BP,CS:New_CG                ; Set pointer to new character generator
     RET
Make_Bold ENDP

;************************************************************************
;                                                                       *
; Make_Italics                                                          *
;    Convert 8xN character genertor to italics, by shifting each byte   *
;    to the right for top, and left for bottom two bytes.               *
;                                                                       *
; Entry: DL -   Number of bytes in each character                       *
;     ES:BP - Pointer to the character generator                        *
;                                                                       *
;************************************************************************

Make_Italics   PROC NEAR

     ;Setup counters

     MOV  BL,DL                       ;Set counter of bytes
     XOR  BH,BH
     MOV  DX,256                      ;Set counter of characters
     MOV  AX,ES                       ;Set pointer to source
     MOV  DS,AX
     MOV  SI,BP
     LEA  DI,CS:New_CG                ;Set pointer to destination
     MOV  AX,CS
     MOV  ES,AX

     ; Loop over characters to change

MI_Char_Loop:
     MOV  CX,BX                       ;Set counter of bytes
     REP  MOVSB                       ;Copy next character
     SUB  DI,BX                       ;Point at first byte
     SHR  BYTE PTR ES:[DI],1          ;Shift top two lines to the right
     SHR  BYTE PTR ES:[DI+1],1
     SHR  BYTE PTR ES:[DI+2],1
     SHR  BYTE PTR ES:[DI+3],1
     SHL  BYTE PTR ES:[DI][BX-1],1    ;Shift last two lines to the left
     SHL  BYTE PTR ES:[DI][BX-2],1
     SHL  BYTE PTR ES:[DI][BX-3],1
     SHL  BYTE PTR ES:[DI][BX-4],1
     ADD  DI,BX                       ;Point to next character
     DEC  DX                          ;Check if all chars done
     JG   MI_Char_Loop
     MOV  DL,BL                       ;Restore DL

     ; Clean up and exit

     LEA  BP,CS:New_CG                ; Set pointer to new character generator
     RET
Make_Italics   ENDP

;************************************************************************
;                                                                       *
; Make_Inverted                                                         *
;    Convert 8xN character genertor to inverse, by inverting each       *
;    byte of the original.                                              *
;                                                                       *
; Entry: DL -   Number of bytes in each character                       *
;     ES:BP - Pointer to the character generator                        *
;                                                                       *
;************************************************************************
```

```
Make_Inverted  PROC NEAR

     ;Setup counters

     MOV  BX,256                       ;Set counter of characters
     XOR  CH,CH                        ;Set counter of bytes
     MOV  AX,ES                        ;Set pointer to source
     MOV  DS,AX
     MOV  SI,BP
     LEA  DI,CS:New_CG                 ;Set pointer to destination
     MOV  AX,CS
     MOV  ES,AX

     ; Loop over characters to change

MV_Char_Loop:
     MOV  CL,DL                        ;Set counter of bytes

     ; Loop over bytes to change

MV_Byte_Loop:
     LODSB                             ;Fetch original byte
     NOT  AL                           ;Invert the byte
     STOSB                             ;Save new character
     LOOP MV_Byte_Loop                 ;Check if all bytes done
     DEC  BX
     JG   MV_Char_Loop                 ;Check if all chars done

     ; Clean up and exit

     LEA  BP,CS:New_CG                 ; Set pointer to new character generator
     RET
Make_Inverted  ENDP

;**************************************************************************
;                                                                         *
; Load_CG                                                                 *
;    Load character generator into plane 2 at the given offset.          *
; Entry:                                                                  *
;    DI   - Offset of character generator in plane 2                     *
;    ES:BP     - Pointer to character generator                           *
;    DL   - Height of each character                                      *
;                                                                         *
;**************************************************************************

Load_CG PROC   NEAR

     ; Enable plane 2 for write at A0000

     MOV  BX,DX                        ; Save character height
     MOV  DX,SEQUENCER_PORT            ; Address of sequencer
     MOV  AL,PLANE_ENABLE_REG          ; Plane enable reg index
     OUT  DX,AL                        ; Select register
     INC  DX
     IN   AL,DX                        ; Fetch current value
     PUSH AX                           ; Save to be restored at the end
     MOV  AL,4                         ; Select plane 2
     OUT  DX,AL

     DEC  DX
     MOV  AL,4                         ; Memory mode reg index
     OUT  DX,AL                        ; Select memory mode registers
     INC  DX
     IN   AL,DX                        ; Fetch current value
     PUSH AX                           ; Save to be restored later
     OR   AL,04h                       ; Disable odd/even
     OUT  DX,AL

     MOV  DX,GRAPHICS_CTRL_PORT        ; Address of graphics controller
     MOV  AL,MISC_REG                  ; Index of misc reg
     OUT  DX,AL                        ; Select misc reg
```

```
        INC  DX
        IN   AL,DX                      ; Read current value
        PUSH AX                         ; Save to be restored later
        AND  AL,0F1h                    ; Disable odd/even and select A000
        OR   AL,04h
        OUT  DX,AL

        ; Copy character generator

        MOV  AX,ES                      ; Load DS:SI with source
        MOV  DS,AX
        MOV  SI,BP
        MOV  AX,0A000h                  ; Load ES:DI with destination
        MOV  ES,AX
        MOV  DX,BX                      ; Setup counters
        XOR  DH,DH
        MOV  BX,256
Loop1:
        MOV  CX,DX                      ; Number of bytes to copy
        REP  MOVSB                      ; Copy bytes for next character
        MOV  CX,20h                     ; Number of zero's to fill after char
        SUB  CX,DX
        REP  STOSB                      ; Fill trailing zeros
        DEC  BX                         ; Check if all characters done
        JG   Loop1                      ; Go do next char, if not all done

        ; Restore previous state

        MOV  DX,GRAPHICS_CTRL_PORT      ; Restore graphics controller
        POP  AX                         ; Get the original value
        XCHG AL,AH                      ; Setup index and data
        MOV  AL,MISC_REG
        OUT  DX,AX                      ; Restore register

        MOV  DX,SEQUENCER_PORT          ; Restore graphics controller
        POP  AX                         ; Get the original value
        XCHG AL,AH                      ; Setup index and data
        MOV  AL,4
        OUT  DX,AX                      ; Restore register

        POP  AX                         ; Get the original value
        XCHG AL,AH                      ; Setup index and data
        MOV  AL,PLANE_ENABLE_REG
        OUT  DX,AX                      ; Restore register

        RET
Load_CG ENDP

;*************************************************************************
; Data definition                                                        *
;*************************************************************************

Title_Msg       DB   '8 SIMULTANEOUS FONTS'
MSG_0           DB   '0: Default Set    '
MSG_1           DB   '2: 8x14 Bold      '
MSG_2           DB   '4: 8x14 Italics   '
MSG_3           DB   '6: 8x14 Inverted  '
MSG_4           DB   '1: 8x8 Normal     '
MSG_5           DB   '7: 8x8 Bold       '
MSG_6           DB   '5: 8x8 Italics    '
MSG_7           DB   '3: 8x8 Inverted   '

MSG_Ptr         DW   OFFSET MSG_0
                DW   OFFSET MSG_1
                DW   OFFSET MSG_2
                DW   OFFSET MSG_3
                DW   OFFSET MSG_4
                DW   OFFSET MSG_5
                DW   OFFSET MSG_6
                DW   OFFSET MSG_7
```

```
_TEXT       ENDS

_STACK      SEGMENT PARA STACK 'STACK'
            DB    100h DUP(?)
_STACK      ENDS
            END
```

## Detection and Identification

Tseng Labs does not have a recommended way of detecting the presence of their product in the system. The Segment Select register at I/O address 3CDh, used for page selection, can be used to detect the ET3000 chip. To verify the presence of the ET3000, code similar to the following can be used:

```
      MOV          DX,3CDh                ;Address of page select register
      IN           AL,DX                  ;Read current value
      MOV          AH,AL                  ;Save for later
      AND          AL,0C0h                ;Preserve msb bits
      OR           AL,55h                 ;Test value one
      OUT          DX,AL                  ;Write test value
      IN           AL,DX                  ;Read value just written out
      CMP          AL,55h                 ;Same value read back?
      JNE          Not_Tseng3000          ;...No, cannot be Tseng ET3000
      MOV          AL,0AAh                ;Test value two
      OUT          DX,AL                  ;Write second test value
      IN           AL,DX                  ;Read back test value
      CMP          AL,AAh                 ;Same value read back?
      JNE          Not_Tseng3000          ;...No, cannot be Tseng ET3000
Tseng3000_Found:
```

# 18

# Western Digital WD90C00 Western Digital Paradise VGA 1024

PARADISE™ © WDC '89 WD90C00

# Introduction

In 1986 Paradise Systems was acquired by Western Digital; Paradise systems is now a trademark of Western Digital Corporation. Initially, chips were still labeled as manufactured by Paradise Systems. VGA chips were originally labeled as PVGA1A with the Paradise logo.

Later chips started to appear with the WD90C00 label (corresponding to the PVGA1B chip) and with the Western Digital Logo. Western Digital now manufactures all their own VGA chips, preserving the Paradise name only for the marketing of VGA boards. Western Digital also builds VGA boards for sale to large OEM customers, including Hewlett-Packard, who resell them under their own brand names. Western Digital claims that there are more Paradise VGA chips in use than chips from any other manufacturer, including IBM.

Paradise VGA 1024 includes up to 512K of display memory and supports resolutions up to 1024x768 with 16 colors or 640x480 with 256 colors. It also includes emulation modes for EGA, CGA, MDA and Hercules compatibility and 132-column text modes. Video output is analog only (TTL displays are not supported).

Unless stated otherwise the information in this chapter applies both to the Paradise VGA 1024 and the Paradise VGA Professional. VGA Professional includes 512K of display memory and has capabilities similar to VGA 1024, except that 1024x768 modes are not supported.

Paradise provides application drivers for many popular applications such as MS-Windows, GEM, AutoDesk products, Ventura Publisher, Cadvance, Framework II, Generic CADD, Lotus and Symphony, OS/2 PM, VersaCAD, Wordperfect and Wordstar 3.3. New drivers are continually added and are available on the Western Digital BBS system.

# AT and Micro Channel Versions

Western Digital offers two different implementations of their VGA boards, an AT bus version and a Micro Channel bus version. These two versions differ in several respects:

- Hardware interrupts are not supported on the AT bus versions.
- Micro Channel versions allow switching between color and monochrome display modes. AT versions will follow the BIOS equipment flag to determine which display modes are allowed (monochrome or color). This allows Paradise VGAs to coexist with a secondary adapter (CGA or MDA) in AT systems.
- The board can be disabled on Micro Channel versions via port 3C3h. On AT versions this port is at address 46E8h (which is also redundantly mapped at 56E8h, 66E8h and 76E8h).

- AT versions display a reverse video intensified character as white on white (the character disappears). On Micro Channel versions characters with such attributes are visible. This can be selected via a switch on Paradise VGA boards.

# New Display Modes

Table 18-1 lists the enhanced display modes that are supported. Any of these modes can be selected by issuing a BIOS mode select command.

**Table 18-1. Enhanced modes—Paradise VGA 1024**

| Mode | Type | Resolution | Colors | Memory Required | Display Type |
|---|---|---|---|---|---|
| 54h | Text | 132 col x 43 rows | 16 | 256 KB | VGA |
| 55h | Text | 132 col x 25 rows | 16 | 256 KB | VGA |
| 56h | Text | 132 col x 43 rows | Mono | 256 KB | VGA |
| 57h | Text | 132 col x 25 rows | Mono | 256 KB | VGA |
| 58h | Graphics | 800x600 | 16 | 256 KB | Multi |
| 59h (2) | Graphics | 800x600 | 2 | 256 KB | Multi |
| 5Ah (1)(2) | Graphics | 1024x768 | 2 | 256 KB | 8514 |
| 5Bh (1) | Graphics | 1024x768 | 4 | 256 KB | 8514 |
| 5Dh (1) | Graphics | 1024x768 | 16 | 512 KB | 8514 |
| 5Ch (3) | Graphics | 800x600 | 256 | 512 KB | Multi |
| 5Eh | Graphics | 640x400 | 256 | 256 KB | VGA |
| 5Fh | Graphics | 640x480 | 256 | 512 KB | VGA |

Note (1): Modes 5Ah, 5Bh and 5Dh are not supported on Professional VGA.
Note (2): Modes 56h, 57h, 59h and 5Ah use CRTC address 3B4h, 3B5h.
Note (3): Mode 5C is not supported on boards manufactured before June 1990

# Memory Organization

For all extended display modes of the VGA 1024, display memory organization is closely patterned after standard IBM VGA display modes.

VGA 1024 includes a display memory paging mechanism that is needed in some display modes to make the entire display memory accessible to the processor. Display memory paging is described in detail later in this chapter.

## High Resolution Text Modes

These modes utilize memory maps that are similar to those used in standard text modes (modes 0,1,2,3 and 7), except that the number of characters per line, or number

of lines per screen, is increased. Display memory is organized as shown in Figure 5-1 (see Chapter 5).

## 2-Color Graphics Modes 59h and 5Ah

Memory organization for these modes resembles VGA mode 11h (640x480 2-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is similar to that shown in Figure 7-1, except that only one plane is used for each byte (plane 0 for even bytes, plane 1 for odd bytes). Care must taken to leave both planes 0 and 1 enabled for writing during drawing operations.

## 4-Color Graphics Mode 5Bh

Memory organization for mode 5Bh resembles VGA modes 4 and 5 (320x200 2-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased, and no memory interleave is used. Display memory organization is linear; each byte contains four pixels, with the topmost bits D6 and D7 corresponding to the left-most pixel. Display memory organization is shown in Figure 18-1. To learn more about this mode, see "Packed Pixels" in Chapter 9.

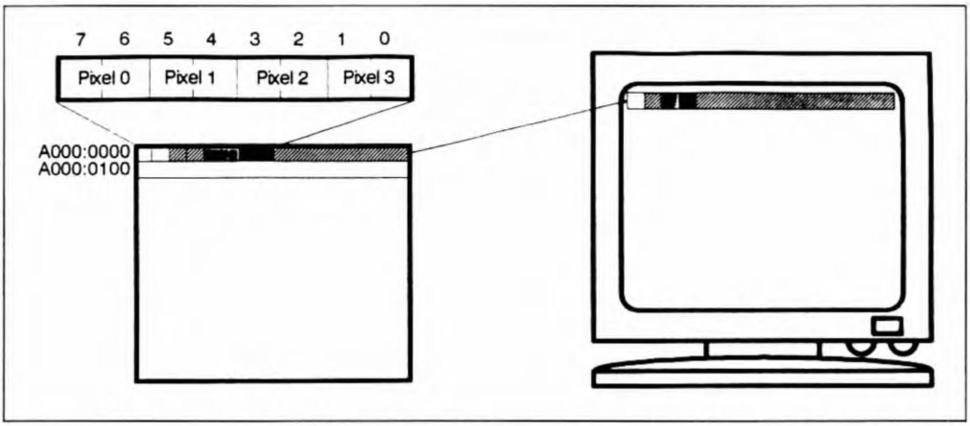


**Figure 18-1. Display memory—mode 5Bh**

## 16-Color Graphics Modes

Memory organization for these modes resembles VGA mode 12h (640x480 16-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is shown in Figure 7-1. See chapter 7 for programming examples.

## 256-Color Graphics Modes

Memory organization for these modes resembles VGA mode 13h (320x200 256-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is shown in Figure 8-1. See chapter 8 for programming examples.

# New Registers

Table 18-2 on page 450 lists the new registers that have been added to the PVGA chip. Some of these are IBM standard registers for interfacing with Micro Channel; others control extended functions of the PVGA. Extended registers are mapped at previously unused indexes of the Graphics Controller (3CEh and 3CFh). PVGA1B contains additional extended registers that are mapped at previously unused indexes of the CRT Controller (3B4h/3D4h and 3B5h/3D5h).

Table 18-2. Extended registers—Paradise PVGA1A, PVGA1B (WD90C00)

| Address | Index | Description |
|---|---|---|
| 46E8h (1) | | Module disable (AT version) - Write only |
| 94h (2) | | Setup register (Micro Channel version) - Write only |
| 3C3h (2) | | Module disable (Micro Channel version) - Write only |
| 102h | | VGA Sleep register (setup mode only) |
| 3CEh,3CFh | 09h | PR0A: Address Offset A (paging control) |
| 3CEh,3CFh | 0Ah | PR0B: Address Offset B (paging control) |
| 3CEh,3CFh | 0Bh | PR1: Memory Size |
| 3CEh,3CFh | 0Ch | PR2: Video Select - do not modify |
| 3CEh,3CFh | 0Dh | PR3: CRT Control - do not modify |
| 3CEh,3CFh | 0Eh | PR4: Video Control - do not modify |
| 3CEh,3CFh | 0Fh | PR5: Extended Register Locking (index 09h to 0Fh) |
| WD90C00(PVGA1B) only: | | |
| 3x4h,3x5h | 29h | PR10: Extended Register Locking (index 2Ah to 30h) |
| 3x4h,3x5h | 2Ah | PR11: EGA Switches |
| 3x4h,3x5h | 2Bh | PR12: Scratch Pad |
| 3x4h,3x5h | 2Ch | PR13: Interlace H/2 Start |
| 3x4h,3x5h | 2Dh | PR14: Interlace H/2 End |
| 3x4h,3x5h | 2Eh | PR15: Miscellaneous Control 1 |
| 3x4h,3x5h | 2Fh | PR16: Miscellaneous Control 2 |
| 3x4h,3x5h | 30h | PR17: Miscellaneous Control 3 |

Note (1): This register is available only on AT versions (instead of 3C3h).
Note (2): This register is available only on Micro Channel versions (instead of 46E8h).

On the VGA Professional, extended registers are locked after a mode change and are left locked by subsequent BIOS calls; on the VGA 1024 extended registers are unlocked after mode change and are left unlocked by subsequent BIOS calls.

# Module Disable (I/O Address 46E8H, 56E8H, 66E8H, 76E8H)

D7-D5 - Unused
D4 - Setup
D3 - Enable I/O and memory accesses
D2-D0 - BIOS ROM page select (8 pages of 4K each)

This write-only register is redundantly addressable at four different I/O addresses. It is used to enable or disable the VGA, and also for BIOS ROM page selection.

Paradise followed IBM's lead by implementing a paging scheme for the VGA BIOS ROM on PS/2 systems. The implementation is slightly different than IBM's, however. In IBM PS/2 systems, the VGA BIOS ROM initializes on reset to page zero. In the Paradise VGA Plus, VGA Plus 16, VGA Professional, and the VGA 1024, it initializes at page six. In

the VGA Plus 16, VGA Professional, and VGA 1024, writing a 0 to this register will select page six; on the VGA Plus it will select page zero.

Western Digital has stated that future revisions of their product will not employ ROM paging.

Setup (bit D4) is initialized on reset to 0, which is the normal operating mode for VGA. When this bit is set to 1, the VGA is placed in setup mode. When in setup mode, all accesses to the VGA, except for port 102h and 46E8h, are disabled.

## POS Sleep Bit Register (I/O Address 102h)

D7-D1 - reserved
D0 - VGA sleep

IBM uses this port on both the AT and Micro Channel VGAs. When set to one, the VGA is enabled for access; when set to 0, VGA is disabled. This register can be accessed only in setup mode.

## Extended Register Bank (I/O Addresses 3CEh and 3CFh, 3B4h/3D4h and 3B5h/3D5h)

Several locking mechanisms are employed for the extended register bank to insure that emulation modes will operate properly. The extended register bank is initially write protected; to access these registers, the unlock registers 3CEh index 0Fh and 3x4 index 29h must be loaded with a code of XXXXX101 binary. Registers 3CEh index 09h through index 0Fh cannot be accessed if the EGA emulation bit (D1) is set in the Video Control register (3CEh, index 0Eh). Registers 3B4/3D4 index 2Ah through index 30h cannot be accessed unless register 3B4/3D4 index 29h is set to a binary value of 1XXX0XXX (X = don't care).

## Address Offset A (I/O Address 3CEh Index 9)

Paradise VGA chips include a flexible and powerful mechanism for display memory paging. Two completely independent memory pages are supported, each with read and write capability, with varying size and granularity. The page size is either 32K or 64K and the granularity is 4K (see Display Memory Paging in Chapter 5 for more details on granularity and page size).

Although the Paradise paging scheme can be used to improve some drawing algorithms, the examples in this book assume 64K pages with 64K granularity (with the exception of BITBLT.ASM, which includes an example of a block transfer when two fully independent 32K pages are available).

Address Offset Register defines the base address of the first page of display memory; in other words, it defines what section of display memory will be accessible to the host in page A. This register contains a seven-bit value which is added to CPU address bits A12 through A18 to access display memory. By default (when dual paging is disabled) page A starts at A000:0 and addresses a 64K window.

Dual paging is enabled by setting bit D3 of the the Memory Size register (index 0Bh) to one. Page A then starts at A800:0, addresses a 32K window, and must be set to the desired page number minus 8. To select page 17 for page A, for example, while dual paging is enabled, first load a 9 (17 - 8 = 9) into Address Offset register A. This is illustrated in Figures 18-2 and 18-3, and in programming examples given later in this chapter.

Memory page size is determined by page enable, and by the host window size as indicated in Table 18-3. To learn more about this register, see the programming examples later in this chapter.

**Table 18-3. Display memory page addresses**

| 3CEh Index 0Bh Bit 3 | 3CEh Index 6 bits 3&2 | Host Address Window | Page Size and Start Address Page A | Page B |
|---|---|---|---|---|
| 0 | 0 0 | A000:0 - BFFF:F | 64kB, A000:0 | disabled |
| 0 | 0 1 | A000:0 - AFFF:F | 64kB, A000:0 | disabled |
| 0 | 1 0 | B000:0 - B7FF:F | 64kB, B000:0 | disabled |
| 0 | 0 0 | B800:0 - BFFF:F | 64kB, B800:0 | disabled |
| 1 | 0 0 | A000:0 - BFFF:F | 64kB, B000:0 | 64kB, A000:0 |
| 1 | 0 1 | A000:0 - AFFF:F | 32kB, A800:0 | 32kB, A000:0 |
| 1 | 1 0 | B000:0 - B7FF:F | 32kB, B800:0 | 32kB, B000:0 |
| 1 | 0 0 | B800:0 - BFFF:F | invalid | invalid |

## Address Offset B (I/O Address 3CEh Index 0Ah)

D7 - unused
D6-D0 - Address offset

Address Offset Register B is used to select a second display memory page (page B). This register contains a 7-bit value that is added to memory address bits A12 through

A18 during processor reads and writes to display memory. Normally, when dual paging is enabled, page B starts at A000:0 and can access 32K. The starting address and page size can be affected by changing the host address window, as shown in Table 18-3. To learn more about paging, see the programming examples later in this chapter.

## Memory Size (I/O Address 3CEh Index 0Bh)

D7-D4 - Memory Size - Do not modify
D3 - Enable Alternate Address Offset
D2 - Enable 16-bit interface to display RAM (1 = enabled)
D1 - Enable 16-bit interface to BIOS ROM (1 = enabled)
D0 - Disable BIOS ROM (1 = disabled)

Enable Alternate Address Offset (Dual Page Enable) allows two pages of display memory to be accessed simultaneously at two different host memory addresses. This is extremely useful when display data must be moved from one page of display memory to another, which is frequently the case during BITBLT operations. Address Offset registers (index 9 and Ah) define what section of display memory will be accessible to the host at each page. The Miscellaneous register of the Graphics Controller defines what host addresses each page will be mapped at (see Table 18-3). To learn more about paging, see the programming examples later in this chapter.

## Video Select (I/O Address 3CEh Index 0Ch)

D7 - AT&T/M24 Mode Enable, 400 line enable
D6 - 6845 Compatibility (0: VGA or EGA, 1: 6845)
D5 - Character Map Select
D4,D3 - Character Clock Period Control
D2 - Character Map Select/Underline
D1 - Third Clock Select Line VCLK2
D0 - Force VCLK (overrides SEQ1 bit 3)

Character Map Select (D5 and D2), with bit D3 of the character attribute, enable character maps from planes 2 or 3 to be selected as follows:

| D5 | D2 | Attribute | Plane Select |
|---|---|---|---|
| 0 | 0 | X | 2 |
| 0 | 1 | X | 2 |
| 1 | 0 | X | 3 |
| 1 | 1 | 0 | 2 |
| 1 | 1 | 1 | 3 |

Selecting page mode addressing (by setting register 3B4h/3D4h index 2Eh bit D2 to one) overrides the plane select table shown above.

Character Clock Period Control determines the width of characters in text modes as follows:

| D4,D3 | Character Width |
|---|---|
| 00 | IBM VGA character clock (8 or 9 dots) |
| 01 | 7 dots (used for 132 character mode) |
| 10 | 9 dots |
| 11 | 10 dots |

Selecting 10 dots per character modifies the function of the horizontal PEL Panning register (Address 3C0h index 13h). The following values should be used for horizontal panning:

| PEL Panning Register Value | PELS Shifted Left |
|---|---|
| 09 | 0 |
| 08 | 1 |
| 00 | 2 |
| 01 | 3 |
| 02 | 4 |
| 03 | 5 |
| 04 | 6 |
| 05 | 7 |
| 06 | 8 |
| 07 | 9 |

When the Underline and Character Map Select/Underline bit (D2) is set to one, character attribute bit D0 will cause a character to be underlined. This overrides the background color function of attribute bit D3, allowing only eight choices of background color. With the Character Map Select bit (D5), this bit is also decoded to enable character maps from planes 2 or 3. See Character Map Select, bit D5, for details.

The Third Clock bit is the third clock select line (VCLK2) to the clock generator chip.

Force VCLK forces the horizontal sync timing clock of the CRT Controller to VCLK. This is for compatibility modes that require locking of the CRTC timing parameters.

## CRT Lock Control (I/O Address 3CEh Index 0Dh)

D7 - Lock VSYNC Polarity
D6 - Lock HSYNC Polarity
D5 - Lock Horizontal Timing

D4 - Bit 9 Control
D3 - Bit 8 Control
D2 - CRTC Control
D1 - Lock Prevention
D0 - Lock Vertical Timing

Register locking is controlled by 4 bits: D0, D1 and D5 of this register and bit D7 of the Vertical Retrace End register (3B5/3D5 index 11).

- **Lock Horizontal Timing** (D5) locks registers associated with horizontal timing.
- **Lock Vertical Timing** (D0) locks registers associated with vertical timing.
- **Lock VSYNC polarity** (D7) locks the polarity of vertical sync.
- **Lock HSYNC polarity** (D6) locks the polarity of horizontal sync.
- **Bit 8 Control** (D3) locks bit 8 of the Start Memory Address High and Cursor Location High registers of the CRT Controller.
- **CRTC Control** (D2) multiplies the Cursor Start, Cursor Stop, Preset Row Scan, and Maximum Scan Line registers by two.
- **Lock Prevention** (D1) inhibits the locking of registers through the Vertical Retrace End register.

## Video Control (I/O Address 3CEh Index 0Eh)

D7 - BLNK/Display Enable
D6 - PCLK = VCLK
D5 - Tri-state Video Outputs
D4 - Tri-state Memory Control Outputs
D3 - Override CGA Enable Video bit
D2 - Lock Internal Palette and Overscan registers
D1 - EGA Compatibility
D0 - Extended 256 color Shift Register Control

**Override CGA Enable Video** (D3) overrides the CGA "enable video" bit D3 of CGA Mode register 3D8 in CGA text mode.

**Lock Internal Palette and Overscan** (D2) locks the palette and overscan registers.

**EGA Compatibility** (D1) disables reads to all registers which are write-only on the IBM EGA, and to the extended registers PR0-PR5.

**Extended 256-color Shift Register Control** (D0) configures the video shift registers for extended 256-color mode.

## General Purpose Status Bits (Address 3CEh, Index 0Fh)

D7 - Read CNF(7) Status
D6 - Read CNF(6) Status
D5 - Read CNF(5) Status
D4 - Read CNF(4) Status
D3 - Read CNF(8) Status
D2-D0 - PR0-PR4 Unlock

Bits D2-D0, when set to 5, will unlock extended registers 3CEh index 9 through index 0Eh. This register also reads back configuration register bits D4 through D8. Setting bit D4 to 1 read protects registers 3CEh index 9 through index 0Eh.

## Unlock Second Bank (I/O Address 3B4h/3D4h Index 29h)

A second bank of extended registers is available in the PVGA1B (WD90C00). This register is used to enable and disable access to this bank.

D7 - Read Enable Bit 1
D6-D4 - Scratch Pad
D3 - Read Enable Bit 0
D2-D0 - Write Enable

**Write Enable** (D2-D0) must be set to 5 (101b) to write enable the register bank. **Read Enable** bits D7 and D3 must be set to 1 and 0 respectively to read enable registers the register bank. A code of 85h written to this register will read and write enable the register bank.

## EGA Switches (I/O Address 3B4h/3D4h Index 2Ah)

| | |
|---|---|
| 7 | EGA Switch 4 |
| 6 | EGA Switch 3 |
| 5 | EGA Switch 2 |
| 4 | EGA Switch 1 |
| 3 | EGA Emulation on Analog Display |
| 2 | Lock Clock Select |
| 1 | Lock Graphics and Sequencer Screen Control |
| 0 | Lock 8/9 Dot Character Clock |

**EGA Configuration switches** (D7-D4) are both readable and writable and are latched at reset to the settings of the on-board switches. These bits can be read on bit D4 of port 3C2h if EGA compatibility mode is enabled.

**Lock Graphics Controller/Sequencer screen control** (D1) inhibits write access to the following bits in the Graphics Controller and Sequencer:

| | |
|---|---|
| Graphics Controller | 3CFh index 5 bits D5 and D6 |
| Sequencer | 3C5h index 1 bits D2-D5 |
| Sequencer | 3C5h index 3 bits D0-D5 |

**Lock 8/9 dots** (D0) inhibits write access to Sequencer register 3C5h index 1, bit D0.

## Scratch Pad (I/O Address 3B4h/3D4h Index 2Bh)

D7 to D0 - Scratch Pad

The data in this register is unaffected by hardware reset and undefined at power up. This register is used by the BIOS and should not be changed.

## Interlace H/2 Start (I/O Address 3B4h/3D4h Index 2Ch)

D7 to D0 - Interlaced H/2 Start

The data in this register is unaffected by hardware reset and undefined at power up. This register defines the starting horizontal character count at which vertical timing is clocked on alternate fields in interlaced operation. The register value should be determined as follows:

Interlaced H/2 start = HORIZ_RETRACE_START - (HORIZ_TOTAL + 5)/2 + HRD
HRD = Horizontal Retrace Delay, D5,D6 of Horizontal Retrace End Register

## Interlace H/2 End (I/O Address 3B4h/3D4h Index 2Dh)

D7 - Enable IRQ
D6 - Vertical Double Scan for EGA on PS/2 Display
D5 - Enable Interlaced Mode
D4-D0 - Interlaced H/2 Start

**Enable IRQ** (D7) enables vertical retrace interrupts on the AT bus. This bit cannot be used in Micro Channel systems.

**Vertical double scan** (D6) is used when emulating EGA on a PS/2 display. Setting this bit causes the Vertical Displayed line counter and row scan counter of the CRT Controller to be clocked by divide-by-two horizontal timing if the vertical sync polarity (3C2 Bit 7=0) is programmed to be positive. The relationship between the actual number of lines displayed [N] and the data [n] programmed into the Vertical Display Enable End register becomes:

$$N = 2(n + 1)$$

And likewise for the actual number of scan lines per character row [N] and the data [n] programmed in the maximum Scan Line register.

**Enable Interlaced Mode** (D5) selects interlaced mode. The Maximum Scan Line register must be set to 0XX00000. Line compare and double scan are not supported in interlaced mode.

**Interlaced H/2 End** (D4-D0) adjusts horizontal sync width for interlaced mode.

## Miscellaneous Control 1 (I/O Address 3B4h/3D4h Index 2Eh)

D7 - Read 46E8H Enable
D6 - Low VCLK
D5 - VCLK1,VCLK2 Latched Outputs
D4 - VCLK = MCLK
D3 - 8514/A Interlaced Compatibility
D2 - Enable Page Mode
D1 - Select Display Enable Timing
D0 - Disable Border

**Read 46E8H Enable** (D7) enables I/O port 46E8H to be read in AT bus systems. Only bits D0-D4 of port 46E8H are readable.

**Low VCLK** (D6) adjusts memory timing to allow a video clock (VCLK) frequency which is much lower than the memory clock (MCLK) frequency. This bit should be set to 1 if the following expression is satisfied:

(MCLK in MHZ) / (VCLK in MHZ) > 2

**Latched VCLK1 and VCLK2** (D5) is used only if VCLK1 and VCLK2 are configured as outputs. It causes outputs VCLK1 and VCLK2 to equal bits D2 and D3 of the Miscellaneous Output register (3C2h).

**VCLK = MCLK** (D4) causes the MCLK input to be selected for the source of all video timing. The other three VCLK inputs cannot be selected when this bit is set.

**Interlaced Compatibility** (D3) should be set to one if exact 8514/A video timing is required. It causes vertical sync to be generated from the trailing edge horizontal sync instead of the leading edge. Interlaced mode must be enabled.

**Enable Page Mode Addressing** (D2) forces screen refresh memory cycles to use page mode addressing in text modes. Page mode addressing is automatically used in graphics modes.

This bit will alter the use of the Character Map Select register as shown below.

| Char Map Select D4 | D3 | Char. Attr. D3 | Plane Selected |
|---|---|---|---|
| 0 | 0 | X | 2 |
| 1 | 1 | X | 3 |
| 1 | 0 | 0 | 2 |
| 1 | 0 | 1 | 3 |
| 0 | 1 | 0 | 3 |
| 0 | 1 | 1 | 2 |

If it is used, this bit must be set before loading the character maps into video RAM, since the addressing of page mode character maps differs from their addressing in nonpage mode. This bit is automatically set by the BIOS in standard 132-column text modes.

**Disable Border** (D0) forces the video outputs to 0 during the interval when border (overscan) color would be active.

## Miscellaneous Control 3 (I/O Address 3B4h/3D4h Index 30h)

D7 to D1 - Reserved
D0 - Map out 2K in BIOS ROM

This bit disables access to the BIOS ROM in the system address range C600:0H - C67F:FH to allow VGA to coexist with adapters such as the IBM PGC which uses this space. This bit is set by a hardware reset.

# The BIOS

VGA boards based on the PVGA1A chip use a BIOS memory space from C000:0h to C000:5FFFh, leaving the 2K region from C000:6000h to C000:7FFFh available for compatibility with adapters such as the IBM Professional Graphics Controller (PGC). VGA 1024 boards using the WD90C00 VGA chip consume this additional 2K address space.

BIOS function 0, mode select, can be used to select any of the Paradise extended display modes. In addition, Paradise has added new subfunctions to BIOS function 0 for BIOS versions -014 and later.

## Parametric Mode Set

**Input Parameters:**

AH = 0
AL = 7Eh

BX = Horizontal resolution (in pixels for graphics modes, character columns for text)
CX = Vertical resolution (in pixels for graphics modes, character rows for text)
DX = Number of colors (0 for monochrome)

**Return Value: None.**

This mode set will only work for standard Paradise display modes.

## Enable/Disable Emulation Mode

**Input Parameters:**

AH = 0
AL = 7Fh
BH = 0 to disable emulation, 1 to enable emulation

**Return Value:**

BH = 7Fh if successful

If the current display mode is 0,1,2,3,4,5 or 6, this call will enable or disable CGA emulation. If the current display mode is 7, it will enable or disable MDA/Hercules emulation.

## Inquire Emulation Status

**Input Parameters:**

AH = 0
AL = 7Fh
BH = 2

**Return Value:**

BH = 7Fh if successful
BL = 1 if emulation is on, 0 if emulation is off
CH = Size of display memory (in units of 64 kbytes)
CL = Memory required by current mode (in units of 64 kbytes)

## Lock Emulation Mode for Reset

**Input Parameters:**

AH = 0

AL = 7Fh
BH = 3

**Return Value:**

BH = 7Fh if successful

## Enable MDA/Hercules Emulation

**Input Parameters:**

AH = 0
AL = 7Fh
BH = 4

**Return Value:**

BH = 7Fh if successful

## Enable CGA Emulation

**Input Parameters:**

AH = 0
AL = 7Fh
BH = 5

**Return Value:**

BH = 7Fh if successful

## Set Monochrome VGA Mode

**Input Parameters:**

AH = 0
AL = 7Fh
BH = 6

**Return Value:**

BH = 7Fh if successful

## Set Color VGA Mode

**Input Parameters:**

AH = 0
AL = 7Fh
BH = 7

**Return Value:**

BH = 7Fh if successful

## Read Paradise Extended Register

**Input Parameters:**

AH = 0
AL = 7Fh
BH = 10h + register index (from port 3CFh)

**Return Value:**

BH = 7Fh if successful
BL = Register value

## Write Paradise Extended Register

**Input Parameters:**

AH = 0
AL = 7Fh
BH = Register index (from port 3CFh)
BL = Data value to write to register

**Return Value:**

BH = 7Fh if successful

Register index 0fh (Locking register) cannot be modified using this function.

## Set Hardware EGA Emulation

**Input Parameters:**

AH = 0

AL = 7Fh
BH = 20h
BL = EGA switch combination (monitor type)

**Return Value:**

BH = 7Fh if successful

# Programming Examples

## Accessing Extended Registers

When writing to extended registers, either one 16-bit instruction **(OUT DX,AX)** or two 8-bit instructions **(OUT DX,AL)** can be used.

After mode select, all extended registers are locked on the VGA Professional (except the Extended Register Locking register, index 0Fh). To enable access to the extended registers, a value of 5 must be written to the Extended Register Locking register bits D0-D2. Code similar to the following can be used to enable access:

```
MOV     DX,3CEh         ;Address of extended block
MOV     AX,050Fh        ;Index Fh and value 5
OUT     DX,AX           ;Enable extended register access
```

Further examples showing access to the extended register bank can be found in the procedures Select_Page, Select_Read_Page and Select_Write_Page shown below.

Many drawing algorithms (especially for sixteen-color modes) can be implemented efficiently by using a 'moving bit mask' while drawing partial bytes. In such algorithms, the BIT MASK register index in the Graphics Controller may be selected at the start of the algorithm:

```
MOV     DX,3CEh
MOV     AL,8
OUT     DX,AL
INC     DX
```

After the index is set, only the data register need be accessed in the inner drawing loop:

```
ROL     AL,1            ;Compute new mask
OUT     DX,AL           ;Enable bits for write using mask
```

Since Paradise paging registers reside at the same I/O address as the Graphics Controller, care must be taken to ensure that after a new page is selected the previous register index is restored so that these drawing algorithms will operate properly.

# Display Memory Paging

The display memory paging mechanism of the Paradise VGA maps selected portions of the display memory to the processor. Operation of display memory paging is very similar to the paging mechanism used for expanded memory boards (also called EMS or LIM memory). A 64K or 128K logical page of VGA RAM (a chunk of display memory) is mapped into the PC host address space in the normal VGA display memory address space.

Either one or two display memory pages may be enabled. Unlike many other VGA products, both memory pages are readable and writable. This can be very useful when transferring data from one part of display memory to another (BITBLT). Display memory paging is illustrated in Figures 18-2 and 18-3. To learn more about paging on Paradise boards, see the description of **Address Offset Registers** earlier in this chapter.

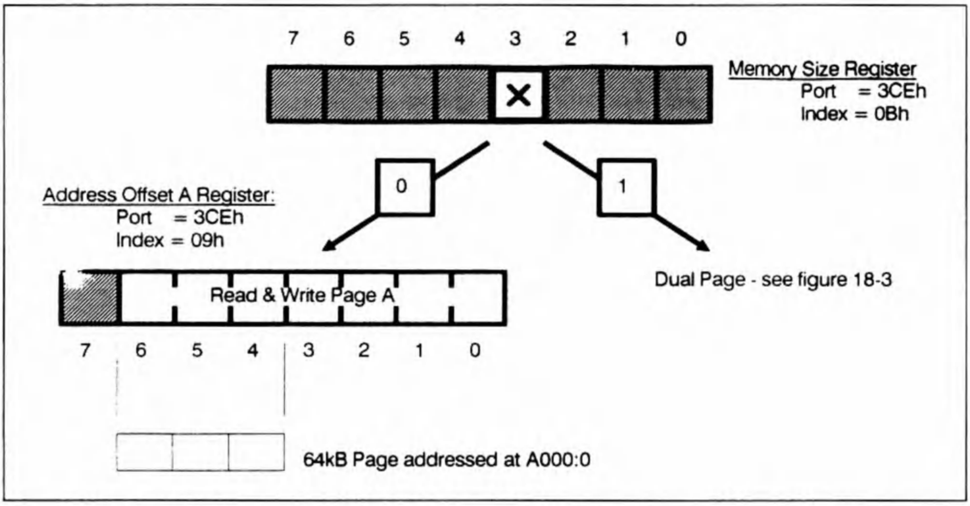


Figure 18-2. Memory paging—single page

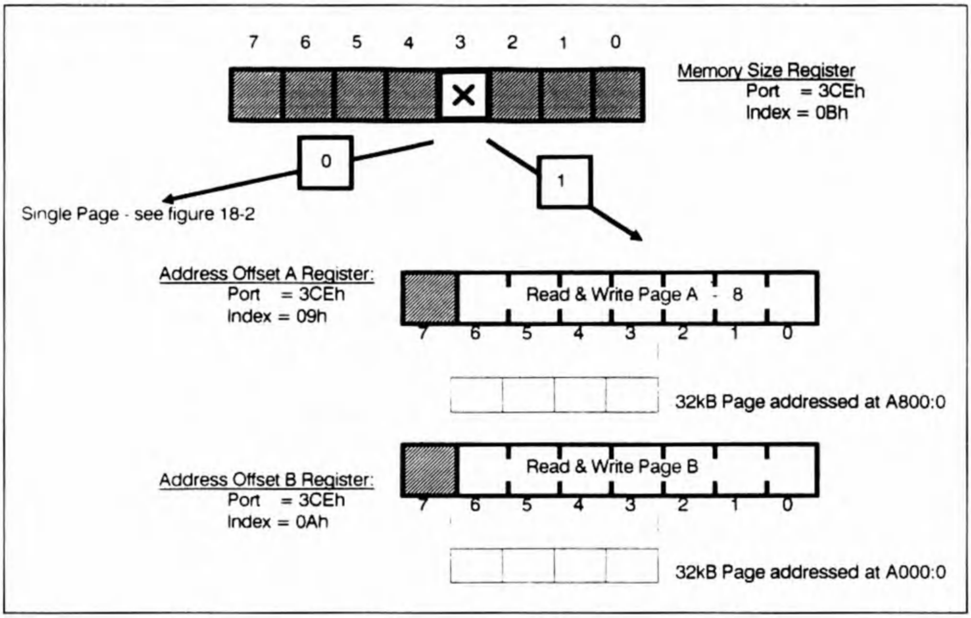


**Figure 18-3. Memory paging—two pages**

Listing 18-1 illustrates how the paging registers are used. Note that the extended register bank is enabled in procedure _Select_Graphics, and disabled in procedure _Select_Text. To easily interface with our common drawing routines, the paging routines in Listing 18-1 do not take full advantage of the Paradise paging capabilities. Select_Page assumes that a 64K page has been requested, and the page is converted to a Paradise page address.

Select_Read_Page and Select_Write_Page assume that a 32K page has been requested with granularity of 32K. Select_Write_Page and Select_Read_Page assume that the read page is in page A, addressed by DS at A800, and the write page is in page B, addressed by ES at A000h. Note that in _Select_Read_Page the page number is adjusted by eight.

All three paging routines preserve the index of the Graphics Controller.

## Listing 18-1. File: WD\SELECT.ASM

```
;*************************************************************************
;* File:        SELECT.ASM                                               *
;* Description: This module contains procedures to select mode and to   *
;*              select pages.  It also initializes global variables      *
;*              according to the values in the MODE.INC include file.    *
;* Entry Points:                                                         *
;*              _Select_Graphics   - Select a graphics mode              *
;*              _Select_Text       - Set VGA adapter into text mode      *
;*              _Select_Page       - Select 64k page                     *
;*              _Select_Read_Page  - Select 32k page A                   *
;*              _Select_Write_Page - Select 32k page B                   *
;* Uses:                                                                 *
;*              MODE.INC           - Mode dependent constants            *
;*              Following are modes and paths for Paradise 1024:         *
;*       |----- 256 colors ------|  |-- 16 colors --| 4 colors 2 colors *
;*      640x400  640x480  800x600  800x600  1024x768 1024x768 1024x768 *
;* Mode:   5Eh      5Fh      N/A      58h       5Dh      5Bh     5Ah    *
;* Path: 256COL   256COL     N/A     16COL     16COL  4COLPACK  2COL    *
;*************************************************************************

        INCLUDE VGA.INC
        INCLUDE MODE.INC        ;Mode dependent constants

        PUBLIC  _Select_Graphics
        PUBLIC  _Select_Text
        PUBLIC  _Select_Page
        PUBLIC  _Select_Read_Page
        PUBLIC  _Select_Write_Page

        PUBLIC  Select_Page
        PUBLIC  Select_Read_Page
        PUBLIC  Select_Write_Page
        PUBLIC  Enable_Dual_Page
        PUBLIC  Disable_Dual_Page

        PUBLIC  Graf_Seg
        PUBLIC  Video_Height
        PUBLIC  Video_Width
        PUBLIC  Video_Pitch
        PUBLIC  Video_Pages
        PUBLIC  Ras_Buffer
        PUBLIC  Two_Pages

        PUBLIC  Last_Byte

;-------------------------------------------------------------------------
; Data segment variables
;-------------------------------------------------------------------------

;_DATA   SEGMENT WORD PUBLIC 'DATA'
;_DATA   ENDS

;-------------------------------------------------------------------------
; Constant definitions
;-------------------------------------------------------------------------

EXTEND_REG_ADDR EQU     3CEh            ;IO Address for page select register
UNLOCK_REG      EQU     00Fh            ;Index for lock/unlock register
PAGEA_REG       EQU     009h            ;Index for page A
PAGEB_REG       EQU     00Ah            ;Index for page B
DUAL_ENABLE_REG EQU     00Bh            ;Index for dual page enable

;-------------------------------------------------------------------------
; Code segment variables
;-------------------------------------------------------------------------

_TEXT   SEGMENT BYTE PUBLIC 'CODE'
Graf_Seg        DW      0A000h          ;Graphics segment addresses
```

```
                DW      0A800h
OffScreen_Seg   DW      0A000h          ;First byte beyond visible screen
Video_Pitch     DW      SCREEN_PITCH    ;Number of bytes in one raster
Video_Height    DW      SCREEN_HEIGHT   ;Number of rasters
Video_Width     DW      SCREEN_WIDTH    ;Number of pixels in a raster
Video_Pages     DW      SCREEN_PAGES    ;Number of pages in the screen
Ras_Buffer      DB      1024 DUP (0)    ;Working buffer
R_Page          DB      0FFh            ;Most recently selected page
W_Page          DB      0FFh
RW_Page         DB      0FFh
Two_Pages       DB      CAN_DO_RW       ;Indicate separate R & W capability

;*************************************************************************
;*                                                                       *
;* _Select_Graphics(HorizPtr, VertPtr, ColorsPtr)                        *
;*      Initialize VGA adapter to 640x400 mode with                      *
;*      256 colors.                                                      *
;*                                                                       *
;* Entry:                                                                *
;*      None                                                             *
;*                                                                       *
;* Returns:                                                              *
;*      VertPtr   - Vertical resolution                                  *
;*      HorizPtr  - Horizontal resolution                                *
;*      ColorsPtr - Number of supported colors                           *
;*                                                                       *
;*************************************************************************

Arg_HorizPtr    EQU     WORD PTR [BP+4] ;Formal parameters
Arg_VertPtr     EQU     WORD PTR [BP+6] ;Formal parameters
Arg_ColorsPtr   EQU     WORD PTR [BP+8] ;Formal parameters

_Select_Graphics PROC NEAR
        PUSH    BP                      ;Standard C entry point
        MOV     BP,SP

        PUSH    DI                      ;Preserve segment registers
        PUSH    SI
        PUSH    DS
        PUSH    ES

        ; Select graphics mode

        MOV     AX,GRAPHICS_ MODE       ;Select graphics mode
        INT     10h

        ; Reset 'last selected page'

        MOV     AL,0FFh                 ;Use 'non-existent' page number
        MOV     CS:R_Page,AL            ;Set currently selected page
        MOV     CS:W_Page,AL
        MOV     CS:RW_Page,AL

        ; Set return parameters

        MOV     SI,Arg_VertPtr          ;Fetch pointer to vertical resolution
        MOV     WORD PTR [SI],SCREEN_HEIGHT     ;Set vertical resolution
        MOV     SI,Arg_HorizPtr         ;Fetch pointer to horizontal resolution
        MOV     WORD PTR [SI],SCREEN_WIDTH      ;Set horizontal resolution
        MOV     SI,Arg_ColorsPtr        ;Fetch pointer to number of colors
        MOV     WORD PTR [SI],SCREEN_COLORS     ;Set number of colors

        ; Enable extended register access

        MOV     DX,EXTEND_REG_ADDR      ;Fetch address of extended reg bank
        MOV     AX,UNLOCK_REG+0500h     ;Unlock extended registers
        OUT     DX,AX

        ; Clean up and return to caller

        POP     ES                      ;Restore segment registers
```

```
        POP     DS
        POP     SI
        POP     DI

        MOV     SP,BP                   ;Standard C exit point
        POP     BP
        RET
_Select_Graphics ENDP

;************************************************************************
;                                                                       *
; Select_Page                                                           *
; Entry:                                                                *
;       AL - Page number                                                *
;                                                                       *
;************************************************************************

Select_Page     PROC NEAR
        CMP     AL,CS:RW_Page           ;Check if already selected
        JNE     SP_Go
        RET
SP_Go:
        PUSH    AX
        PUSH    BX
        PUSH    DX
        AND     AL,7                    ;Force into range
        MOV     CS:RW_Page,AL           ;Save as most recent RW page
        MOV     CS:R_Page,0FFh          ;Invalidate R and W pages
        MOV     CS:W_Page,0FFh

        ; Preserve index
        ; 16 color drawing routines assume that mask register of gr. ctrlr
        ; remains select.  Since paging uses same I/O address, we must
        ; preserve the index.

        MOV     DX,EXTEND_REG_ADDR      ;Save graphics controller index
        IN      AL,DX
        MOV     BL,AL

        ; Select next page

        MOV     AH,CS:RW_Page           ;Fetch page number
        SHL     AH,1                    ;Convert 64k multiple to 4k mult
        SHL     AH,1
        SHL     AH,1
        SHL     AH,1
        MOV     AL,PAGEA_REG            ;Select page
        OUT     DX,AX

        ; Restore index

        MOV     AL,BL                   ;Restore index
        OUT     DX,AL

        POP     DX
        POP     BX
        POP     AX
        RET
Select_Page     ENDP

;************************************************************************
;                                                                       *
; Select_Read_Page                                                      *
; Entry:                                                                *
;       AL - Page number                                                *
;                                                                       *
;************************************************************************

Select_Read_Page PROC NEAR
        CMP     AL,CS:R_Page            ;Check if already selected
        JNE     SRP_Go
```

```
        RET
SRP_Go:
        PUSH    AX
        PUSH    BX
        PUSH    DX
        AND     AL,0Fh                  ;Force into range
        MOV     CS:RW_Page,0FFh         ;Invalidate RW page number
        MOV     CS:R_Page,AL            ;Save as most recently selected

        ; Preserve index
        ; 16 color drawing routines assume that mask register of gr. ctrlr
        ; remains select.  Since paging uses same I/O address, we must
        ; preserve the index

        MOV     DX,EXTEND_REG_ADDR      ;Save graphics controller index
        IN      AL,DX
        MOV     BL,AL

        ; Select next page

        MOV     AH,CS:R_ Page           ;Fetch page number
        SHL     AH,1                    ;Convert 32k multiple to 4k mult
        SHL     AH,1
        SHL     AH,1
        SUB     AH,08h                  ;Adjust for ES=A800 instead of A000
        MOV     AL,PAGEA_REG            ;Select page
        OUT     DX,AX

        ; Restore index

        MOV     AL,BL                   ;Restore index
        OUT     DX,AL

        POP     DX
        POP     BX
        POP     AX

        RET
Select_Read_Page ENDP

;*********************************************************************
;                                                                    *
; Select_Write_Page                                                  *
; Entry:                                                             *
;       AL - Page number                                             *
;                                                                    *
;*********************************************************************

Select_Write_Page PROC NEAR
        CMP     AL,CS:W_Page            ;Check if already selected
        JNE     SWP_Go
        RET
SWP_Go:
        PUSH    AX
        PUSH    BX
        PUSH    DX
        AND     AL,0Fh                  ;Force into range
        MOV     CS:RW_Page,0FFh         ;Invalidate RW page number
        MOV     CS:W_Page,AL            ;Save as most recently selected

        ; Preserve index
        ; 16 color drawing routines assume that mask register of gr. ctrlr
        ; remains select.  Since paging uses same I/O address, we must
        ; preserve the index

        MOV     DX,EXTEND_REG_ADDR      ;Save graphics controller index
        IN      AL,DX
        MOV     BL,AL

        ; Select next page
```

```
        MOV     AH,CS:W_ Page           ;Fetch page number
        SHL     AH,1                    ;Convert 32k multiple to 4k mult
        SHL     AH,1
        SHL     AH,1
        MOV     AL,PAGEB_REG            ;Select page
        OUT     DX,AX

        ; Restore index

        MOV     AL,BL                   ;Restore index
        OUT     DX,AL

        POP     DX
        POP     BX
        POP     AX

        RET
Select_Write_Page ENDP

;*********************************************************************
;                                                                    *
; Enable_Dual_Page                                                   *
; Disable_Dual_Page                                                  *
;                                                                    *
; Entry:                                                             *
;       AL - Page number                                             *
;                                                                    *
;*********************************************************************

Enable_Dual_Page PROC NEAR
        PUSH    AX
        PUSH    DX
        MOV     DX,EXTEND_REG_ADDR      ;Fetch address of exteneded reg block
        MOV     AL,DUAL_ENABLE_REG      ;Fetch index of dual enable
        OUT     DX,AL                   ;Select register
        INC     DX
        IN      AL,DX                   ;Read previous value
        OR      AL,08h                  ;Set enable bit
        OUT     DX,AL                   ;Enable dual paging
        POP     DX
        POP     AX
        RET
Enable_Dual_Page ENDP

Disable_Dual_Page PROC NEAR
        PUSH    AX
        PUSH    DX
        MOV     DX,EXTEND_REG_ADDR      ;Fetch address of exteneded reg block
        MOV     AL,DUAL_ENABLE_REG      ;Fetch index of dual enable
        OUT     DX,AL                   ;Select register
        INC     DX
        IN      AL,DX                   ;Read previous value
        AND     AL,NOT 08h              ;Clear enable bit
        OUT     DX,AL                   ;Enable dual paging
        POP     DX
        POP     AX
        RET
Disable_Dual_Page ENDP

;*********************************************************************
;                                                                    *
; _Select_Page(PageNumber)                                           *
; Entry:                                                             *
;       PageNumber - Page number                                     *
;                                                                    *
;*********************************************************************

Arg_PageNumber  EQU     BYTE PTR [BP+4]

_Select_Page    PROC NEAR
        PUSH    BP                      ;Setup frame pointer
```

```
        MOV     SP,BP
        MOV     AL,Arg_PageNumber       ;Fetch argument
        POP     BP                      ;Restore BP
        JMP     Select_Page
_Select_Page    ENDP

;*************************************************************************
;                                                                        *
; _Select_Read_Page(PageNumber)                                          *
; Entry:                                                                 *
;       PageNumber- Page number for read                                 *
;                                                                        *
;*************************************************************************

Arg_PageNumber  EQU     BYTE PTR [BP+4]

_Select_Read_Page       PROC NEAR
        PUSH    BP                      ;Setup frame pointer
        MOV     SP,BP
        MOV     AL,Arg_PageNumber       ;Fetch argument
        POP     BP                      ;Restore BP
        JMP     Select_Read_Page
_Select_Read_Page       ENDP

;*************************************************************************
;                                                                        *
; _Select_Write_Page(PageNumber)                                         *
; Entry:                                                                 *
;       PageNumber - Page number for write                               *
;                                                                        *
;*************************************************************************

Arg_PageNumber  EQU     BYTE PTR [BP+4]

_Select_Write_Page      PROC NEAR
        PUSH    BP                      ;Setup frame pointer
        MOV     SP,BP
        MOV     AL,Arg_PageNumber       ;Fetch argument
        POP     BP                      ;Restore BP
        JMP     Select_Write_Page
_Select_Write_Page      ENDP

;*************************************************************************
;*                                                                       *
;* _Select_Text                                                          *
;*      Set VGA adapter to text mode                                     *
;*                                                                       *
;*************************************************************************

_Select_Text    PROC NEAR
        MOV     AX,TEXT_MODE            ;Select mode 3
        INT     10h                     ;Use BIOS to reset mode
        RET
_Select_Text    ENDP

Last_Byte:
_Text   ENDS
        END
```

## BITBLT with Two Pages

Paradise is one of the few VGAs that is capable of supporting two separate read and write pages. A more efficient method for block copying (BITBLT) can be used. An example of such an improved algorithm can be found at the end of the file BITBLT.ASM in Chapter 7.

# Detection and Identification

Paradise recommends that the presence of a Paradise BIOS is detected by checking for the ASCII string 'VGA=' at BIOS ROM location C000:007Dh. Code similar to the following can be used to check for the Paradise BIOS.

```
                ; Check for Paradise BIOS
                MOV             AX,C000h                    ;Fetch segment of BIOS
                MOV             DS,AX
                MOV             SI,7Dh                      ;Fetch offset of signature
                CMP             WORD PTR [SI],4756h         ;Check for first half of signature
                JNE             Not_Paradise_BIOS
                CMP             WORD PTR [SI+2],3D41h       ;Check for second half of signature
                JNE             Not_Paradise_BIOS
Paradise_BIOS_Found:                                        ;We found eternal bliss....
```

To detect boards based on PVGA1A and WD90C00 chips, the locking mechanism of the extended registers can be used. Locked and unlocked extended registers can be written and read to detect which VGA chip version is present. The PVGA1A does not contain a second bank of extended registers. Code similar to that shown below can be used.

In this code it is assumed that the board is in a standard VGA mode and that both unlock registers are initially readable (otherwise they may not be restored properly). All registers are restored to their original values at the end of the test.

```
                ;Save current value of Lock/Unlock register
Look_For_PVGA1A:
                MOV             DX,3CEh                     ;Address of extended register bank 1
                MOV             AL,0Fh                      ;Index of Unlock register
                OUT             DX,AL                       ;Select Unlock register
                INC             DX
                IN              AL,DX                       ;Read current value
                MOV             BL,AL                       ;Save current value for later
                ;Unlock extended register bank 1
                MOV             AL,05h                      ;Value to use for unlock function
                OUT             DX,AL                       ;Unlock extended register bank 1
                ;Save current content of page register A
                MOV             AL,09h                      ;Index of page A register
                OUT             DX,AL                       ;Select register
                INC             DX
                IN              AL,DX                       ;Read current value
                MOV             BH,AL                       ;Save current value for later
                ; Write first pattern to page A and read it back
                MOV             AL,05h                      ;Pattern to write
                OUT             DX,AL                       ;Write first pattern
                XOR             AL,AL
                IN              AL,DX                       ;Read back
                CMP             AL,05h                      ;Verify value read back
                JNE             Not_Paradise                ;Quit if read not same as written
                ; Write second pattern and read it back
                MOV             AL,0Ah                      ;Pattern to write
                OUT             DX,AL                       ;Write first pattern
                XOR             AL,AL
                IN              AL,DX                       ;Read back
                CMP             AL,0Ah                      ;Verify value read back
                JNE             Not_Paradise                ;Quit if read not same as written
                ;Lock paging register
                MOV             DX,3CEh                     ;Address of extended register bank 1
                MOV             AL,0Fh                      ;Index of Unlock register
                MOV             AH,00h                      ;Value to use for lock function
```

```
                OUT     DX,AX               ;Lock extended register bank 1
                ; Write first pattern to page A and read it back
                MOV     AL,09h              ;Index of page register
                OUT     DX,AL               ;Select page register
                INC     DX
                MOV     AL,05h              ;Pattern to write
                OUT     DX,AL               ;Write first pattern
                XOR     AL,AL
                IN      AL,DX               ;Read back
                CMP     AL,05h              ;Verify value read back
                JE      Not_Paradise        ;Quit if read same as written
                ;Restore original values
                MOV     AL,BH               ;Fetch original value of paging reg
                OUT     DX,AL               ;Restore paging register
                DEC     DX                  ;Address
                MOV     AL,09h              ;Index of Unlock reg
                MOV     AH,BL               ;Original value of Unlock reg
                OUT     DX,AX               ;Restore Unlock register
Paradise_Found:                             ;...and what a bliss...
                ;Unlock Scratch Pad register for read and write
Look_For_WD90C00:
                MOV     AX,0                ;Point to BIOS data area
                MOV     ES,AX               ;   to fetch address
                MOV     SI,463h             ;... of the CRT controller
                MOV     DX,[SI]             ;Fetch address of extended bank 2
                MOV     AL,29h              ;Index of Unlock register
                OUT     DX,AL               ;Select Unlock register
                INC     DX
                IN      AL,DX               ;Read current value
                MOV     BL,AL               ;Save current value for later
                MOV     AL,85h              ;Value to use for unlock function
                OUT     DX,AL               ;Unlock extended register bank 2
                ;Save current content of scratch register
                MOV     AL,2Bh              ;Index of scratch pad
                OUT     DX,AL               ;Select register
                INC     DX
                IN      AL,DX               ;Read current value
                MOV     BH,AL               ;Save current value for later
                ; Write first pattern and read it back
                MOV     AL,05h              ;Pattern to write
                OUT     DX,AL               ;Write first pattern
                XOR     AL,AL
                IN      AL,DX               ;Read back
                CMP     AL,05h              ;Verify value read back
                JNE     PVGA1A_Found        ;Quit if read not same as written
                ; Write second pattern and read it back
                MOV     AL,0Ah              ;Pattern to write
                OUT     DX,AL               ;Write first pattern
                XOR     AL,AL
                IN      AL,DX               ;Read back
                CMP     AL,0Ah              ;Verify value read back
                JNE     PVGA1A_Found        ;Quit if read not same as written
                ;Restore original values
                MOV     AL,BH               ;Fetch original value
                OUT     DX,AL               ;Restore scratch pad register
                DEC     DX                  ;Address
                MOV     AL,29h              ;Index of Unlock reg
                MOV     AH,BL               ;Original value of Unlock reg
                OUT     DX,AX               ;Restore Unlock register
                ;Do processing for WD90C00
WD90C00_Found:
                ...
                ;Do processing for PVGA1A
PVGA1A_Found:
                ...                         ;Restore original values
                ;Do processing for non-Paradise board
Not_Paradise:
                                            ;Restore original values
```

# 19

# ZyMOS Poach 51 TrueTech HiRes VGA

# Introduction

TrueTech manufactures VGA products that are based on the ZyMOS POACH51 VGA chip. ZyMOS POACH51 is an equivalent (second-source) part to the Trident 8800CS VGA chip. As with most SuperVGAs, the POACH51 VGA chip is fully IBM VGA-compatible, includes register level compatibility for EGA, CGA, MDA and Hercules, and includes extended high resolution text and graphics modes. High resolution applications software drivers are also available for products such as AutoCAD, AutoShade, Framework II and III, GEM, Lotus 1-2-3 and Symphony, MS-Windows, Ventura Publisher, and WordPerfect.

# Chip Versions

ZyMOS POACH VGA chips contain a version number that can be read from the least significant nibble of the Hardware Version register (I/O address 3C5h, index 0Bh).

# New Display Modes

Table 19-1 lists the enhanced display modes that are supported by the HiRes VGA. All modes are selectable using the standard BIOS Mode Select call, function 0.

# Memory Organization

For all extended display modes of the HiRes VGA, display memory organization is closely patterned after standard IBM VGA display modes.

For some extended modes, a memory paging mechanism is also used. Memory paging is described in detail later in the programming examples.

## High Resolution Text Modes

These modes utilize memory maps that are similar to those used in standard text modes (modes 0,1,2,3 and 7), except that the number of characters per line and/or number of lines per screen is increased. Display memory is organized as shown in Figure 5-1 (see Chapter 5).

Table 19-1. Enhanced display modes—TrueTech VGA

| Mode | Type | Resolution | Colors | Memory Required | Display Type |
|---|---|---|---|---|---|
| 51h | Text | 80 col x 43 rows | 16 | 256 KB | VGA |
| 52h | Text | 80 col x 60 rows | 16 | 256 KB | VGA |
| 53h | Text | 132 col x 25 rows | 16 | 256 KB | Multi |
| 54h | Text | 100 col x 25 rows | 16 | 256 KB | Multi |
| 55h | Text | 100 col x 60 rows | 16 | 256 KB | Multi |
| 56h | Text | 132 col x 60 rows | 16 | 256 KB | Multi |
| 57h | Text | 132 col x 43 rows | 16 | 256 KB | Multi |
| 5Bh | Graphics | 800x600 | 16 | 256 KB | Multi |
| 5Ch | Graphics | 640x400 | 256 | 256 KB | VGA |
| 5Dh | Graphics | 640x480 | 256 | 512 KB | VGA |
| 5Eh | Graphics | 800x600 | 256 | 512 KB | Multi |
| 5Fh | Graphics | 1024x768 | 16 | 512 KB | 8514 or XL |
| 60h | Graphics | 960x720 | 16 | 512 KB | Multi |
| 61h | Graphics | 1280x640 | 16 | 512 KB | XL |
| 62h | Graphics | 512x512 | 256 | 256 KB | Multi |
| 63h | Graphics | 720x540 | 16 | 256 KB | Multi |
| 64h | Graphics | 720x540 | 256 | 512 KB | Multi |
| 6Ah | Graphics | 800x600 | 16 | 256 KB | Multi |

## 16-Color Graphics Modes

Memory organization for these modes resembles VGA mode 12h (640x480 16-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is shown in Figure 7-1. See Chapter 7 for programming examples.

## 256-Color Graphics Modes

Memory organization for these modes resembles VGA mode 13h (320x200 256-color graphics), except that both the number of pixels per scan line and the number of scan lines are increased. Display memory organization is shown in Figure 8-1. See Chapter 8 for programming examples.

# New Registers

To support the new modes and emulation modes, the ZyMOS chip contains additional registers not found on the standard VGA. These are listed in Table 19-2.

Table 19-2. Extended Registers—ZyMOS POACH51

| Register Name | Address | Index |
|---|---|---|
| CRTC Module Testing register | 3B4h/3D4 | 1Eh |
| Scratch Pad | 3B4h/3D4h | 1Fh |
| Power Up Mode register 1 | 3C4 | 0Ch |
| Power Up Mode register 2 | 3C4 | 0Fh |
| Hardware Version register | 3C4h | 0Bh |
| Mode Control register 1 | 3C4h | 0Eh |
| Mode Control register 2 | 3C4h | 0D |
| CPU Latch Read Back | 3B4h/3D4h | 22h |
| Attribute State Read Back | 3B4h/3D4h | 24h |
| Attribute Index Read Back | 3B4h/3D4h | 26h |
| Video Enable | 3C3h | |
| Display Adapter Enable | 46E8h | |

Registers used in the programming examples, in this text, are described in detail below.

## Hardware Version Register (I/O Address 3C5h Index 0Bh)

D7-D4 - Reserved
D3-D0 - Hardware version

Reading this register causes the chip to enter version 2 paging mode. Writing this register causes the chip to enter version 1 paging mode. Programming examples in this chapter assume version 2 paging. For more details on paging see the Programming Examples in this chapter.

## Mode Control Register 1 (I/O Address 3C5h Index 0Eh)

D7-D4 - Reserved
D3-D0 - 64K page select

This register is used to select a page number in version 2 paging mode. In this mode, bit 1 must be written inverted, but will read back correctly (uninverted). To select page 7, for example, a value of 5 should be written; a value of 7 will be read back.

## Processor Latch Read Back Register (I/O Address 3B5/3D5 Index 22h)

This register can be used to read back the current value of the processor data latch in the Graphics Controller for the color plane that is currently enabled for reading.

## Attribute Controller State Register (I/O Address 3B5/3D5 Index 24h)

D7 - Attribute Controller State (read-only)
D6-D0 - Reserved

**Attribute Controller State** indicates whether the next write operation to the Attribute Controller (I/O address 3C0) will be used as a register index or as register data (0 = index, 1 = data).

## Attribute Controller Index Read Back (I/O Address 3B5/3D5 Index 26h)

This read-only port can be used to read the current value of the index register internal to the Attribute Controller.

# Programming Examples

## Display Memory Paging

Display memory is divided into eight 64K pages. Pages are selected via Mode Control register 1 (I/O address 3C5h, index 0Eh). Bit D1 must be complemented before the page number is written, but will read back uncomplemented. This is illustrated in Figure 19-1. Table 19-3 on page 480 contains a list of valid page numbers and corresponding values.

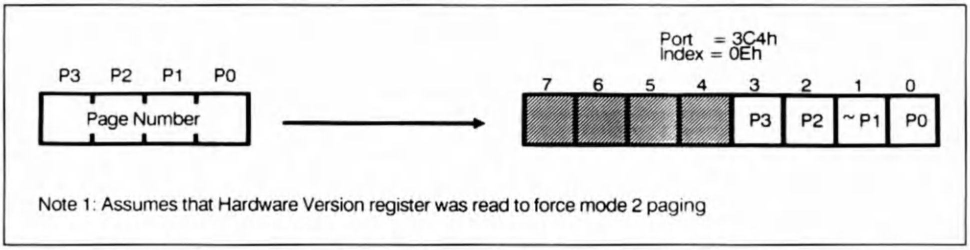


Figure 19-1. ZYMOS Paging register

Access to Mode Control register 1 is enabled by a read operation from the Hardware Version register (I/O address 3C5h, index 0Bh) and is disabled by a write operation to the Hardware Version register. Access to the page select bits in Mode Control register 1 should always be prefaced by a read from the Hardware Version register to assure that it is enabled.

**Table 19-3. Memory paging mode 2**

| | I/O Address 3C5, Index 0Eh | |
|---|---|---|
| | Write Value | Read Value |
| Page Number | D3 D2 D1 D0 | D3 D2 D1 D0 |
| 0 | 0 0 1 0 | 0 0 0 0 |
| 1 | 0 0 1 1 | 0 0 0 1 |
| 2 | 0 0 0 0 | 0 0 1 0 |
| 3 | 0 0 0 1 | 0 0 1 1 |
| 4 | 0 1 1 0 | 0 1 0 0 |
| 5 | 0 1 1 1 | 0 1 0 1 |
| 6 | 0 1 0 0 | 0 1 1 0 |
| 7 | 0 1 0 1 | 0 1 1 1 |

Listing 19-1 contains mode select procedures, Select_Graphics and Select_Text, and the paging procedure Select_Page.

Select_Graphics shows how to invoke extended modes. Note that in procedure Select_Graphics, after the mode select, the 64K page is selected using the Miscellaneous register of the Graphics Controller. Version 2 paging is forced by a read operation from the Hardware Version register. Select_Page shows how to select pages (for version 2 mode).

**Listing 19-1. File: ZYMOS\SELECT.ASM**

```
;*************************************************************************
;* File:  SELECT.ASM                                                     *
;* Description: This module contains procedures to select mode and to   *
;*              select pages.  It also initializes global variables     *
;*              according to the values in the MODE.INC include file.   *
;* Entry Points:                                                         *
;*              _Select_Graphics    - Select a graphics mode             *
;*              _Select_Text        - Set VGA adapter into text mode     *
;*              _Select_Page        - Select page for read and write     *
;* Uses:                                                                 *
;*        MODE.INC      - Mode dependent constants                       *
;*        Following are EXTENED modes and paths for TrueTech             *
;*    |----- 256 colors ------|  |-- 16 colors --| 4 colors 2 colors     *
;*    640x400  640x480  800x600  800x600  1024x768 1024x768  1024x768    *
;* Mode:   5Ch      5Dh      5Eh      6Ah       5Fh      N/A  N/A        *
;* Path: 256COL   256COL   256COL    16COL     16COL     N/A  N/A        *
;*************************************************************************

     INCLUDE VGA.INC
     INCLUDE MODE.INC    ;Mode dependent constants

     PUBLIC    _Select_Graphics
```

```
     PUBLIC    _Select_Text
     PUBLIC    _Select_Page
     PUBLIC    _Select_Read_Page
     PUBLIC    _Select_Write_Page

     PUBLIC    Select_Page
     PUBLIC    Select_Read_Page
     PUBLIC    Select_Write_Page
     PUBLIC    Enable_Dual_Page
     PUBLIC    Disable_Dual_Page

     PUBLIC    Graf_Seg
     PUBLIC    Video_Height
     PUBLIC    Video_Width
     PUBLIC    Video_Pitch
     PUBLIC    Video_Pages
     PUBLIC    Ras_Buffer
     PUBLIC    Two_Pages

     PUBLIC    Last_Byte

;-------------------------------------------------------------------------
; Data segment variables
;-------------------------------------------------------------------------

;_DATA     SEGMENT WORD PUBLIC 'DATA'
;_DATA     ENDS

;-------------------------------------------------------------------------
; Constant definitions
;-------------------------------------------------------------------------

EXTEND_REG_ADDR EQU 3C4h      ;IO Address for extended bank registers
VERSION_REG     EQU 00Bh      ;Index for enable/version register
PAGE_REG        EQU 00Eh      ;Index for page register

;-------------------------------------------------------------------------
; Code segment variables
;-------------------------------------------------------------------------

_TEXT     SEGMENT BYTE PUBLIC 'CODE'

Graf_Seg        DW   0A000h              ;Graphics segment addresses
                DW   0B000h
OffScreen_Seg   DW   0A000h         ;First byte beyond visible screen
Video_Pitch     DW   SCREEN_PITCH   ;Number of bytes in one raster
Video_Height    DW   SCREEN_HEIGHT  ;Number of rasters
Video_Width     DW   SCREEN_WIDTH   ;Number of pixels in a raster
Video_Pages     DW   SCREEN_PAGES   ;Number of pages in the screen
Ras_Buffer      DB   1024 DUP (0)   ;Working buffer
R_Page          DB   0FFh           ;Most recently selected page
W_Page          DB   0FFh
RW_Page         DB   0FFh
Two_Pages       DB   CAN_DO_RW ;Indicate separate R & W capability

;*************************************************************************
;*                                                                       *
;* _Select_Graphics(HorizPtr, VertPtr, ColorsPtr)                        *
;*   Initialize VGA adapter to 640x400 mode with                         *
;*   256 colors.                                                         *
;*                                                                       *
;* Entry:                                                                *
;*   None                                                                *
;*                                                                       *
;* Returns:                                                              *
;*   VertPtr   - Vertical resolution                                     *
;*   HorizPtr  - Horizontal resolution                                   *
;*   ColorsPtr - Number of supported colors                              *
;*                                                                       *
;*************************************************************************
```

```
Arg_HorizPtr    EQU  WORD PTR [BP+4] ;Formal parameters
Arg_VertPtr     EQU  WORD PTR [BP+6] ;Formal parameters
Arg_ColorsPtr   EQU  WORD PTR [BP+8] ;Formal parameters

_Select_Graphics PROC NEAR
     PUSH BP              ;Standard C entry point
     MOV  BP,SP

     PUSH DI              ;Preserve segment registers
     PUSH SI
     PUSH DS
     PUSH ES

     ; Select graphics mode

     MOV  AX,GRAPHICS_MODE     ;Set extended mode number
     INT  10h                  ;Use BIOS to select mode

     ; Reset 'last selected page'

     MOV  AL,0FFh              ;Use 'non-existent' page number
     MOV  CS:R_Page,AL         ;Set currently selected page
     MOV  CS:W_Page,AL
     MOV  CS:RW_Page,AL

     ; Set return parameters

     MOV  SI,Arg_VertPtr       ;Fetch pointer to vertical resolution
     MOV  WORD PTR [SI],SCREEN_HEIGHT   ;Set vertical resolution
     MOV  SI,Arg_HorizPtr      ;Fetch pointer to horizontal resolution
     MOV  WORD PTR [SI],SCREEN_WIDTH    ;Set horizontal resolution
     MOV  SI,Arg_ColorsPtr     ;Fetch pointer to number of colors
     MOV  WORD PTR [SI],SCREEN_COLORS   ;Set number of colors

     ; Enable extended register access for version 2

     MOV  DX,EXTEND_REG_ADDR   ;Address of extended reg bank
     MOV  AL,VERSION_REG       ;Index of version (and enable) reg
     OUT  DX,AL                ;Select register
     INC  DX                   ;Advance to data port
     IN   AL,DX                ;Read version to enable version 2 mode

     MOV  DX,GRAPHICS_CTRL_PORT    ;Address of graphics controller
     MOV  AL,MISC_REG              ;Index of miscellaneous register
     OUT  DX,AL                    ;Select misc register
     INC  DX                       ;Advance to data port
     IN   AL,DX                    ;Read misc register
     AND  AL,0F3h                  ;Clear addressing bits
     OR   AL,04h                   ;Enable A0000-AFFFF addressing
     OUT  DX,AL                    ;Output value

     ; Clean up and return to caller

     POP  ES              ;Restore segment registers
     POP  DS
     POP  SI
     POP  DI

     MOV  SP,BP                ;Standard C exit point
     POP  BP
     RET
_Select_Graphics ENDP
```

```
;*********************************************************************
;                                                                   *
; Select_Page                                                       *
; Entry:                                                            *
;    AL - Page number                                               *
;                                                                   *
;*********************************************************************

Select_Page     PROC NEAR
     CMP        AL,CS:RW_Page    ;Check if already selected
     JNE        SP_Go
     RET
SP_Go:
     PUSH AX
     PUSH DX

     AND  AL,7                  ;Force page number into range
     MOV  CS:RW_Page,AL         ;Save as most recent RW page
     MOV  CS:R_Page,0FFh        ;Invalidate R and W pages
     MOV  CS:W_Page,0FFh
     MOV  AH,AL                 ;Copy page number
     XOR  AH,02h                ;Invert bit 1
     MOV  DX,EXTEND_REG_ADDR    ;Address of extended register bank
     MOV  AL,PAGE_REG           ;Index of select page register
     OUT  DX,AL                 ;Select the page register
     INC  DX                    ;Advance address to data
     IN   AL,DX                 ;Read previous value
     AND  AL,0F0h               ;Preserve upper nibble
     OR   AL,AH                 ;Combine preserved bits with page number
     OUT  DX,AL                 ;Select new page

     POP  DX
     POP  AX
     RET
Select_Page     ENDP

;*********************************************************************
;                                                                   *
; Select_Read_Page                                                  *
; Entry:                                                            *
;    AL - Page number                                               *
;                                                                   *
;*********************************************************************

Select_Read_Page PROC NEAR
     CMP  AL,CS:R_Page          ;Check if already selected
     JNE  SRP_Go
     RET
SRP_Go:
     RET
Select_Read_Page ENDP

;*********************************************************************
;                                                                   *
; Select_Write_Page                                                 *
; Entry:                                                            *
;    AL - Page number                                               *
;                                                                   *
;*********************************************************************

Select_Write_Page PROC NEAR
     CMP  AL,CS:W_Page          ;Check if already selected
     JNE  SWP_Go
     RET
SWP_Go:
     RET
Select_Write_Page ENDP
```

```
;**********************************************************************
;                                                                     *
; Enable_Dual_Page                                                    *
; Disable_Dual_Page                                                   *
;                                                                     *
; Entry:                                                              *
;    AL - Page number                                                 *
;                                                                     *
;**********************************************************************

Enable_Dual_Page PROC NEAR
     RET
Enable_Dual_Page ENDP

Disable_Dual_Page PROC NEAR
     RET
Disable_Dual_Page ENDP

;**********************************************************************
;                                                                     *
; _Select_Page(PageNumber)                                            *
; Entry:                                                              *
;    PageNumber - Page number                                         *
;                                                                     *
;**********************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Page   PROC NEAR
     PUSH BP                   ;Setup frame pointer
     MOV  SP,BP
     MOV  AL,Arg_PageNumber    ;Fetch argument
     POP  BP                   ;Restore BP
     JMP  Select_Page
_Select_Page   ENDP

;**********************************************************************
;                                                                     *
; _Select_Read_Page(PageNumber)                                       *
; Entry:                                                              *
;    PageNumber- Page number for read                                 *
;                                                                     *
;**********************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Read_Page   PROC NEAR
     PUSH BP                   ;Setup frame pointer
     MOV  SP,BP
     MOV  AL,Arg_PageNumber    ;Fetch argument
     POP  BP                   ;Restore BP
     JMP  Select_Read_Page
_Select_Read_Page   ENDP

;**********************************************************************
;                                                                     *
; _Select_Write_Page(PageNumber)                                      *
; Entry:                                                              *
;    PageNumber - Page number for write                               *
;                                                                     *
;**********************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Write_Page  PROC NEAR
     PUSH BP                   ;Setup frame pointer
     MOV  SP,BP
     MOV  AL,Arg_PageNumber    ;Fetch argument
     POP  BP                   ;Restore BP
     JMP  Select_Write_Page
_Select_Write_Page  ENDP
```

```
;***************************************************************************
;                                                                         *
;* _Select_Text                                                           *
;*    Set VGA adapter to text mode                                        *
;                                                                         *
;***************************************************************************

_Select_Text   PROC NEAR
     MOV       AX,TEXT_MODE         ;Select mode 3
     INT       10h                  ;Use BIOS to reset mode
     RET
_Select_Text   ENDP

Last_Byte:
_Text ENDS
      END
```

## Detection and Identification

TrueTech does not have a recommended way to detect the presence of their boards. To detect the ZyMOS VGA chip, we recommend reading the Hardware Version register and checking for a value of 2 in the lower nibble. Code similar to the following can be used:

```
     MOV     DX,3C4h          ;Address of extended reg bank
     MOV     AL,0Bh           ;Index of version register
     OUT     DX,AL            ;Select version register
     INC     DX
     IN      AL,DX            ;Read version
     AND     AL,0Fh           ;Keep only lower nibble
     CMP     AL,2             ;Check for version 2
     JNE     Not_ZyMOS
ZyMOS_Found:
```

This method will not distinguish between the Trident 8800CS chip and ZyMOS POACH51 chip; this should not matter since the devices are register compatible.

# 20

# *The VESA Standard*

VESA
Video Electronics Standards Association

# Introduction

A new industry standards organization known as the Video Electronics Standards Association (VESA) has assumed the task of improving the compatibility of VGA boards from different vendors. VESA membership includes such major VGA suppliers as ATI, Chips and Technologies, Cirrus, Everex, Genoa, Video-Seven, Intel, Orchid, Phoenix Technologies, Sigma Designs, STB, Paradise, and others.

VESA has proposed a set of added BIOS functions that can be used to access the extended modes and capabilities of SuperVGAs in a standard manner. These new BIOS functions are collectively grouped under the new BIOS function 4Fh.

Included in the VESA specification is a standard set of mode numbers for high resolution modes, which is an accomplishment in itself since so many VGA manufacturers have arbitrarily assigned new mode numbers. In order to create standard mode numbers for all current and future extended modes, without conflicting with vendor-specific modes, VESA expanded the size of a mode number from 7 bits to 15 bits. VGA suppliers can continue to support their current mode numbering system while at the same time supporting the VESA standard.

To avoid mode number conflicts, VESA standard mode numbers are greater than or equal to 100h with the exception of mode 6Ah, which is already a defacto industry standard.

# VESA Display Modes

Table 20-1 lists VESA standard modes.

**Table 20-1. VESA standard display modes**

| Mode number | Resolution | Colors |
|---|---|---|
| 100h | 640x400 | 256 |
| 101h | 640x480 | 256 |
| 102h | 800x600 | 16 |
| 103h | 800x600 | 256 |
| 104h | 1024x768 | 16 |
| 105h | 1024x768 | 256 |
| 106h | 1280x1024 | 16 |
| 107h | 1280x1024 | 256 |
| 6Ah(1) | 800x600 | 16 |

Note: Mode 6Ah is the only VESA mode which can be selected using the standard VGA BIOS Mode Select call (function 0). All other modes are selectable using VESA extended BIOS function 2, Set Super VGA Video Mode.

# The VESA BIOS

All VESA extended VGA BIOS functions are accessed using int 10h, as used for standard VGA BIOS functions. The designated Super VGA extended function number is 4Fh. A standard VGA BIOS performs no action for this function number.

All extended VGA BIOS functions have a similar format:

AH = 4Fh
AL = VESA function code (0 through 5)

Every function returns status information in AX. The format of a status word is as follows:

AL = 4Fh if function is supported
AH = 0 if function call was successful
1 if function call failed
2 - FFh reserved (should be treated as failure)

## Function 00h - Return SuperVGA Information

**Input Parameters:**

AH = 4Fh
AL = 00h
ES:DI = Address of destination for information block (256-byte buffer)

**Return Values:**

AL = 4Fh if function is supported
AH = 00h if function was completed successfully

The following information block is returned at the requested address:

```
VESA Signature   db   'VESA'   ;VESA signature
VESA version     db   ?        ;VESA minor version number
                 db   ?        ;VESA major version number
OEM StringPtr    dd   ?        ;pointer to ASCIIZ OEM string
Capabilities     dd   ?        ;board capabilities
VideoModePtr     dd   ?        ;pointer to supported modes
```

**VESA signature** will always be 'VESA'.

**VESA version** will initially be 1.0 (major = 1, minor = 0)

**OEM string pointer** is a pointer to an OEM-defined null terminated string that can be used to identify vendor specific capabilities for hardware specific drivers.

The **Capabilities** field identifies what general features are supported. It is currently unused and should be set to 00000000h.

**Video mode pointer** points to a list of supported SuperVGA modes (both VESA and vendor-specific modes). Each mode number occupies one word (16 bits). The list is terminated by 0FFFFh. The list may be in ROM or in RAM.

## Function 01h - Return SuperVGA Mode Information

**Input Parameters:**

AH = 4Fh
AL = 01h
CX = Desired mode
ES:DI = Address of destination for information block

**Return Values:**

AL = 4Fh if function is supported
AH = 00h if function was completed successfully

The following information block is returned at the requested address:

```
Mode_Attributes       dw   ?   ;mode attributes
Win_A_Attributes      db   ?   ;window A attributes
Win_B_Attributes      db   ?   ;window B attributes
Win_Granularity       dw   ?   ;window granularity
Win_Size              dw   ?   ;window size
Win_A_Segment         dw   ?   ;window A segment address
Win_B_Segment         dw   ?   ;window B segment address
Win_Func_Ptr          dd   ?   ;pointer to window
                               function
Bytes_Per_Scan_Line   dw   ?   ;bytes per scan line
```

Optional information:

```
X_Resolution          dw   ?   ;horizontal resolution
Y_Resolution          dw   ?   ;vertical resolution
X_Char_Size           db   ?   ;character cell width
Y_Char_Size           db   ?   ;character cell height
Number_Of_Planes      db   ?   ;number of memory planes
Bits_Per_Pixel        db   ?   ;bits per pixel
Number_Of_Banks       db   ?   ;number of banks
Memory_Model          db   ?   ;memory model type
Bank_Size             db   ?   ;bank size in kb
```

Optional information fields are not required for standard VESA video modes since these parameters are predefined for each mode.

**Mode_Attributes** describes important characteristics of the display mode:

D15 - D5 - Reserved
D4 - Graphics/Text Mode (1 = graphics)
D3 - Color/Monochrome Mode (1 = color)
D2 - BIOS text functions are supported (1 = true)
D1 - Optional information is valid for block (1 = true)
D0 - Mode is supported by the current display (1 = true)

**Win_A_Attributes** and **Win_B_Attributes** give attributes for display memory windows. In VESA terminology, a window is a page of display memory mapped at a particular address. Either one or two windows may be supported:

D7-D3 - reserved
D2 = 1 if the window is writable
D1 = 1 if the window is readable
D0 = 1 if the window is supported

**Win_Granularity** indicates the smallest increment, in kilobytes, that can be used in selecting the start address for a display memory page.

**Win_Size** specifies the size of a page of display memory (in kilobytes).

**Win_A_Segment** and **Win_B_Segment** specify the host segment address where each display memory window is located.

**Win_Func_Addr** specifies the address of the display memory windowing function. This function can be invoked either through VESA BIOS function 5 or by a direct call to this address. Since speed is usually important in graphics algorithms, a direct call to the routine will probably be the most popular method of accessing this function.

**Bytes_Per_Scan_Line** indicates the logical screen width, which may be equal to or greater than the physical screen width.

**X_Resolution** and **Y_Resolution** specify the width and height of the screen in pixels (for graphics modes) or in characters (for text modes).

**X_Char_Cell_Size** and **Y_Char_Cell_Size** specify the size of a character cell in pixels.

**Memory_Model** indicates the display memory organization used in this mode. Valid types are:

0 - Text mode
1 - CGA graphics
2 - Hercules graphics
3 - Four-plane graphics (see Figure 8-1 on page 181)
4 - Packed pixel graphics (see Figure 7-1 on page 131)

5 - Nonchain 4, 256-color graphics (see Figure 12-1 on page 284)
6-0fh - Reserved by VESA
10h-ffh - May be defined by manufacturer

**Number_Of_Banks** and **Bank_Size** apply only to graphics modes that have nonlinear memory maps such as CGA graphics modes and Hercules graphics modes. Number_Of_Banks indicates the number of logical scan line groupings, and Bank_Size indicates the number of scan lines per group. For more information on these modes, see Chapter 2.

## Function 02h - Set SuperVGA Display Mode

**Input Parameters:**

AH = 4Fh
AL = 02h
BX = display mode number

The **Display mode number** parameter should follow VESA numbering conventions:

D15 - Preserve display memory flag (0: clear memory, 1: preserve memory)
D14 to D9 - Reserved for future expansions (should be 0's)
D8 - VESA mode flag (0: not VESA-defined mode, 1: VESA mode)
D7 to D0 - Mode number (see Table 20-1 for valid VESA defined numbers).

**Return Value:**

AL = 4Fh if function is supported
AH = 00h if function was completed successfully

## Function 03h - Return Current Display Mode

**Input Parameters:**

AH = 4Fh
AL = 03h

**Return Value:**

AL = 4Fh if command was completed successfully
AH = 0 of function was completed successfully
BX = Current display mode

## Function 04h - Save/Restore SuperVGA Video State

This function is actually comprised of three separate subfunctions: Return State Buffer Size, Save SuperVGA Video State, and Restore SuperVGA Video State.

### *Subfunction 1 - Return State Buffer Size*

This function can used to determine the size of the buffer that will be required to save video state information.

**Input Parameters:**

AH = 4Fh
AL = 04h
DL = 0
CX = States to be saved
 D0 - Video hardware state
 D1 - Video BIOS data
 D2 - Video DAC state
 D3 - SuperVGA state

**Return Value:**

AL = 4Fh if function is supported
AH = 00h if function was completed successfully
BX = Number of 64-byte blocks needed to save state

### *Subfunction 2 - Save SuperVGA Video State*

**Input Parameters:**

AH = 4Fh
AL = 04h
DL = 1
CX = States to be saved
 D0 - Video hardware state
 D1 - Video BIOS data
 D2 - Video DAC state
 D3 - SuperVGA state
ES:BX = Pointer to save buffer

**Return Value:**

AL = 4Fh if function is supported
AH = 00h if function was completed successfully

### *Subfunction 3 - Restore SuperVGA State*

**Input Parameters:**

AH = 4Fh
AL = 04h
DL = 2
CX = States to be saved
D0 - Video hardware state
D1 - Video BIOS data
D2 - Video DAC state
D3 - SuperVGA state
ES:BX = pointer to save buffer

**Return Value:**

AL = 4Fh if function is supported
AH = 00h if function was completed successfully

## Function 05h - Display Memory Window Control

This function is needed because of the wide diversity of paging methods used by different manufacturers. It provides a generalized method for selecting a page of display memory, or reading back the current page number.

### *Select Display Memory Page*

**Input Parameters:**

AH = 4Fh
AL = 05h
BH = 0
BL = Window number (0 = window A, 1 = window B)
DX = Page starting boundary (in granularity units)

**Return Value:**

AL = 4Fh if function is supported
AH = 00h if function was completed successfully, 01h otherwise

For faster execution, this function can be called directly with a far call to the address returned by VESA BIOS function 1. Note that the address of the paging function may vary depending on the display mode, or it may not exist (indicated by returned NULL address from function 1).

If the paging function is called directly, registers AL and AH are not needed. No status will be returned and registers AX and DX are destroyed.

To learn more about this function see the programming examples later in this chapter.

### Return Current Display Memory Page

**Input Parameters:**

AH = 4Fh
AL = 05h
BH = 1
BL = Window number (0 = window A, 1 = window B)

**Return Value:**

AL = 4Fh if function is supported
AH = 00h if function was completed successfully
DX = Current page starting boundary (in granularity units)

# Programming Examples

## Display Memory Paging

The VESA BIOS provides two mechanisms for selecting display memory pages. BIOS function 5 may be called to select pages, or for optimum speed function 1 can be used to obtain a far pointer to the paging function, and then the paging function can be called directly.

Listing 20-1 on page 496 shows how to use the first method to select pages. In the procedure _Select_Graphics, VESA BIOS function 01 (Return Super VGA Information) is used to determine if a given mode is supported. If so, the returned information block is examined to determine the type of paging available.

To be consistent with the examples used throughout this book, the VESA programming examples assume 64K pages at address A000:0h. The example in Listing 20-1 may not work properly for modes that use two independent pages, since these typically use two 32K pages placed at different addresses.

This programming example shows how to verify the presence of a VESA BIOS, how to verify support for this mode, how to invoke the mode, and how to select pages.

## Listing 20-1. File: VESA\SELECT.ASM

```
;*************************************************************************
;* File:  SELECT.ASM                                                     *
;* Description: This module contains procedures to selectmode and to    *
;*       select pages.  It also initializes global variables            *
;*       according to the values in the MODE.INC include file.          *
;* Entry Points:                                                         *
;*       _Select_Graphics   - Select a graphics mode                     *
;*       _Select_Text       - Set VGA adapter into text mode             *
;*       _Select_Page       - Select read and write page                 *
;*       _Select_Read_Page  - Select read page only                      *
;*       _Select_Write_Page - Select write page only                     *
;* Uses:                                                                 *
;*       MODE.INC      - Mode dependent constants                        *
;*       Following are modes and paths for VESA BIOS boards:             *
;*    |----- 256 colors ------|  |-- 16 colors --| 4 colors 2 colors     *
;*    640x400  640x480  800x600  800x600  1024x768 1024x768 1024x768     *
;* Mode:   100h    101h     103h      102h     104h      N/A   N/A       *
;* Path:  256COL  256COL   256COL    16COL     16COL     N/A   N/A       *
;*************************************************************************

     INCLUDE VGA.INC
     INCLUDE MODE.INC    ;Mode dependent constants

     PUBLIC    _Select_Graphics
     PUBLIC    _Select_Text
     PUBLIC    _Select_Page
     PUBLIC    _Select_Read_Page
     PUBLIC    _Select_Write_Page

     PUBLIC    Select_Page
     PUBLIC    Select_Read_Page
     PUBLIC    Select_Write_Page
     PUBLIC    Enable_Dual_Page
     PUBLIC    Disable_Dual_Page

     PUBLIC    Graf_Seg
     PUBLIC    Video_Height
     PUBLIC    Video_Width
     PUBLIC    Video_Pitch
     PUBLIC    Video_Pages
     PUBLIC    Ras_Buffer
     PUBLIC    Two_Pages

     PUBLIC    Last_Byte

;-------------------------------------------------------------------------
; Data segment variables
;-------------------------------------------------------------------------

;_DATA     SEGMENT WORD PUBLIC 'DATA'
;_DATA     ENDS

;-------------------------------------------------------------------------
; Constant definitions
;-------------------------------------------------------------------------

ModeInfoStruct STRUC

; Mandatory information (allways provided)

ModeAttributes      dw   ?    ; mode attributes
WinAAttributes      db   ?    ; window A attributes
WinBAttributes      db   ?    ; window B attributes
WinGranularity      dw   ?    ; window granularity
WinSize             dw   ?    ; window size
WinASegment         dw   ?    ; window A start segment
WinBSegment         dw   ?    ; window B start segment
WinFuncPtr          dd   ?    ; pointer to window function
```

```
BytesPerScanLine    dw   ?    ; bytes per scan line

; Optional information (provided if bit D1 of ModeAttributes is set)

XResolution         dw   ?    ; horizontal resolution
YResolution         dw   ?    ; vertical resolution
XCharSize           db   ?    ; character cell width
YCharSize           db   ?    ; character cell height
NumberOfPlanes      db   ?    ; number of memory planes
BitsPerPixel        db   ?    ; bits per pixel
NumberOfBanks       db   ?    ; number of banks
MemoryModel         db   ?    ; memory model type
BankSize            db   ?    ; bank size in kb

ModeInfoStruct ENDS

;-----------------------------------------------------------------------
; Code segment variables
;-----------------------------------------------------------------------

_TEXT     SEGMENT BYTE PUBLIC 'CODE'

Graf_Seg  DW   0A000h              ;Graphics segment addresses
          DW   0A000h
OffScreen_Seg  DW   0A000h        ;First byte beyond visible screen
Video_Pitch    DW   SCREEN_PITCH  ;Number of bytes in one raster
Video_Height   DW   SCREEN_HEIGHT ;Number of rasters
Video_Width    DW   SCREEN_WIDTH  ;Number of pixels in a raster
Video_Pages    DW   SCREEN_PAGES  ;Number of pages in the screen
Ras_Buffer     DB   1024 DUP (0)  ;Working buffer
R_Page         DB   0FFh          ;Most recently selected page
W_Page         DB   0FFh
RW_Page        DB   0FFh
Two_Pages      DB   CAN_DO_RW ;Indicate separate R & W capability
Msg_No_BIOS    DB   '...Error: Cannot locate VESA BIOS',0Dh,0Ah,'$'
Msg_No_Mode    DB   '...Error: Requested mode not supported',0Dh,0Ah,'$'

Mode_Info ModeInfoStruct <>    ;Buffer for mode dependent info

;***********************************************************************
;*                                                                     *
;* _Select_Graphics(HorizPtr, VertPtr, ColorsPtr)                      *
;*   Initialize VGA adapter to 640x400 mode with                       *
;*   256 colors.                                                       *
;*                                                                     *
;* Entry:                                                              *
;*   None                                                              *
;*                                                                     *
;* Returns:                                                            *
;*   VertPtr   - Vertical resolution                                   *
;*   HorizPtr  - Horizontal resolution                                 *
;*   ColorsPtr - Number of supported colors                            *
;*                                                                     *
;***********************************************************************

Arg_HorizPtr   EQU  WORD PTR [BP+4] ;Formal parameters
Arg_VertPtr    EQU  WORD PTR [BP+6] ;Formal parameters
Arg_ColorsPtr  EQU  WORD PTR [BP+8] ;Formal parameters

_Select_Graphics PROC NEAR
     PUSH BP                        ;Standard C entry point
     MOV  BP,SP

     PUSH DI                        ;Preserve segment registers
     PUSH SI
     PUSH DS
     PUSH ES

     ; Verify presence of VESA BIOS

     MOV  AX,CS                     ;Pointer to info buffer
```

```
        MOV   ES,AX
        LEA   DI,CS:Ras_Buffer
        MOV   AX,4F00h                  ;Fn=Return Super VGA Info
        INT   10h
        CMP   AX,004Fh                  ;Check status code
        JNE   Not_VESA_BIOS    ;   and quit if not VESA BIOS
        CMP   WORD PTR ES:[DI],'EV'     ;Check VESA signature in info block
        JNE   Not_VESA_BIOS    ;   and quit if not VESA BIOS
        CMP   WORD PTR ES:[DI+2],'AS'
        JNE   Not_VESA_BIOS
        JMP   VESA_BIOS_Found

Not_VESA_BIOS:
        MOV   AX,CS              ;Pointer to error message
        MOV   DS,AX
        LEA   DX,CS:Msg_No_BIOS
        MOV   AH,09h             ;Fn=Display string on console device
        INT   21h
        MOV   AX,-1              ;Return code = error
        JMP   SG_Done

VESA_BIOS_Found:

        ; Get information about requested mode and select it

        MOV   AX,4F01h           ;Fn=Return Super VGA mode information
        MOV   CX,GRAPHICS_MODE
        LEA   DI,CS:Mode_Info    ;Get pointer to info buffer
        INT   10h                ;Use VESA BIOS to get info about mode
        CMP   AX,004Fh           ;Check if function was successful
        JNE   Mode_Not_Supported ;   and quit if not

        ; To limit number of versions for each drawing routine, only
        ; 64kByte pages are supported.
        ; (And even for 64kByte pages there are already 10 versions.)
        ; Here a check is made that this mode is a 'simple' mode, with
        ; 64k window at A000h.  It is assumed that this mode is
        ; either 16-color planar or 256-color packed pixel organization.

        CMP   ES:[DI].WinSize,64  ;Check that window size is 64k
        JNE   Mode_Not_Supported
        CMP   ES:[DI].WinASegment,0A000h ;Check that window is at A000h
        JNE   Mode_Not_Supported

        MOV   AX,4F02h           ;Fn=Select Super VGA mode
        MOV   BX,GRAPHICS_MODE   ;Mode to select
        INT   10h                ;Select the mode
        CMP   AX,004Fh           ;Check returned status
        JE    Mode_Set

Mode_Not_Supported:

        MOV   AX,CS              ;Pointer to error message
        MOV   DS,AX
        LEA   DX,CS:Msg_No_Mode
        MOV   AH,09h             ;Fn=Display string on console device
        INT   21h
        MOV   AX,-1              ;Return code = error
        JMP   SG_Done

Mode_Set:

        ; Reset 'last selected page'
        MOV   AL,0FFh            ;Use 'non-existent' page number
        MOV   CS:R_Page,AL       ;Set currently selected page
        MOV   CS:W_Page,AL
        MOV   CS:RW_Page,AL

        ; Set return parameters

        MOV   SI,Arg_VertPtr     ;Fetch pointer to vertical resolution
```

```
     MOV  WORD PTR [SI],SCREEN_HEIGHT   ;Set vertical resolution
     MOV  SI,Arg_HorizPtr     ;Fetch pointer to horizontal resolution
     MOV  WORD PTR [SI],SCREEN_WIDTH    ;Set horizontal resolution
     MOV  SI,Arg_ColorsPtr    ;Fetch pointer to number of colors
     MOV  WORD PTR [SI],SCREEN_COLORS   ;Set number of colors

     XOR  AX,AX               ;Return code = success

     ; Clean up and return to caller
SG_Done:
     POP  ES             ;Restore segment registers
     POP  DS
     POP  SI
     POP  DI

     MOV  SP,BP               ;Standard C exit point
     POP  BP
     RET
_Select_Graphics ENDP

;*************************************************************************
;                                                                        *
; Select_Page                                                            *
; Entry:                                                                 *
;    AL - Page number                                                    *
;                                                                        *
;*************************************************************************

Select_Page    PROC NEAR
     CMP  AL,CS:RW_Page       ;Check if already selected
     JNE  SP_Go
     RET
SP_Go:
     PUSH AX
     PUSH BX
     PUSH DX
     MOV  CS:RW_Page,AL       ;Save page number
     ; Convert 64k page number according to board granularity
     MOV  AX,64               ;Assume 64k granule
     XOR  DX,DX
     DIV  CS:Mode_Info.WinGranularity   ;Divide by actual granule size
     MUL  CS:RW_Page          ;Multiply by 64k page number
     ; Select page using VESA BIOS function 05h
     MOV  DX,AX               ;Fetch page number
     MOV  AX,4F05h            ;Fn=Super VGA window control
     MOV  BX,0000h            ;Subfn=Set window, Window=A
     INT  10h                 ;Use VESA BIOS to select page
     ; Cleanup and return
     POP  DX
     POP  BX
     POP  AX
     RET
Select_Page    ENDP

;*************************************************************************
;                                                                        *
; Select_Read_Page                                                       *
;    This function is not supported in this example                      *
; Entry:                                                                 *
;    AL - Page number                                                    *
;                                                                        *
;*************************************************************************

Select_Read_Page PROC NEAR
     RET
Select_Read_Page ENDP
```

```
;*********************************************************************
;                                                                    *
; Select_Write_Page                                                  *
;    This function is not supported in this example                  *
; Entry:                                                             *
;    AL - Page number                                                *
;                                                                    *
;*********************************************************************

Select_Write_Page PROC NEAR
     RET
Select_Write_Page ENDP

;*********************************************************************
;                                                                    *
; _Select_Page(PageNumber)                                           *
; Entry:                                                             *
;    PageNumber - Page number                                        *
;                                                                    *
;*********************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Page   PROC NEAR
     PUSH BP                   ;Setup frame pointer
     MOV  SP,BP
     MOV  AL,Arg_PageNumber    ;Fetch argument
     POP  BP                   ;Restore BP
     JMP  Select_Page
_Select_Page   ENDP

;*********************************************************************
;                                                                    *
; _Select_Read_Page(PageNumber)                                      *
; Entry:                                                             *
;    PageNumber- Page number for read                                *
;                                                                    *
;*********************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Read_Page   PROC NEAR
     PUSH BP                   ;Setup frame pointer
     MOV  SP,BP
     MOV  AL,Arg_PageNumber    ;Fetch argument
     POP  BP                   ;Restore BP
     JMP  Select_Read_Page
_Select_Read_Page   ENDP

;*********************************************************************
;                                                                    *
; _Select_Write_Page(PageNumber)                                     *
; Entry:                                                             *
;    PageNumber - Page number for write                              *
;                                                                    *
;*********************************************************************

Arg_PageNumber EQU  BYTE PTR [BP+4]

_Select_Write_Page  PROC NEAR
     PUSH BP                   ;Setup frame pointer
     MOV  SP,BP
     MOV  AL,Arg_PageNumber    ;Fetch argument
     POP  BP                   ;Restore BP
     JMP  Select_Write_Page
_Select_Write_Page  ENDP
```

```
;**********************************************************************
;*                                                                    *
;* _Select_Text                                                       *
;*   Set VGA adapter to text mode                                     *
;*                                                                    *
;**********************************************************************

_Select_Text   PROC NEAR
     MOV  AX,TEXT_MODE          ;Select mode 3
     INT  10h                   ;Use BIOS to reset mode
     RET
_Select_Text   ENDP

;**********************************************************************
;*                                                                    *
;* Enable_Dual_Page                                                   *
;* Disable_Dual_Page                                                  *
;*                                                                    *
;**********************************************************************

Enable_Dual_Page    PROC NEAR
     RET
Enable_Dual_Page    ENDP

Disable_Dual_Page   PROC NEAR
     RET
Disable_Dual_Page   ENDP

Last_Byte:
_Text     ENDS
          END
```

# Detection and Identification

The Extended VESA BIOS provides a sophisticated detection mechanism that allows a program to determine not only the presence of a VESA BIOS but also a list of modes and information about each mode.

Function 00h, Return SuperVGA Information, can be used to determine the presence of a VESA BIOS from the return status in AX and the signature bytes in the returned information block. Listing 20-2 illustrates how to detect the BIOS VESA and how to determine the list of modes supported.

**Listing 20-2. File: VESA\INFO.C**

```
/**********************************************************************/
/*                                                                    */
/* File:  INFO.C                                                      */
/* Description: This is a program to demonstrate how to test if       */
/*        VESA BIOS is present, and how to use VESA BIOS              */
/*        functions.                                                  */
/*                                                                    */
/**********************************************************************/

#include <stdio.h>
#include <dos.h>
```

```
/************************************************************************/
/* Structure definitions                                                */
/************************************************************************/

/* Board information structure                                          */
struct  {
        char VESASignature[4];  /* 4 signature bytes                    */
        int  VESAVersion;       /* VESA version number                  */
        char far *OEMStringPtr;/* Pointer to OEM string                 */
        char Capabilities[4];  /* Capabilities of the video environment*/
        int far *VideoModePtr; /* Pointer to supported Super VGA modes */
        char    Dummy[246];     /*Info block must be at least 256 Bytes */
        } VESA_Info;

/* Mode information structure                                           */

struct  {
        /* Mandatory information (allways provided)                     */

        int  ModeAttributes;      /* mode attributes                    */
        char WinAAttributes;      /* window A attributes                */
        char WinBAttributes;      /* window B attributes                */
        int  WinGranularity;      /* window granularity                 */
        int  WinSize;             /* window size                        */
        int  WinASegment;         /* window A start segment             */
        int  WinBSegment;         /* window B start segment             */
        char far *WinFuncPtr;     /* pointer to window function         */
        int  BytesPerScanLine;    /* bytes per scan line                */

     /* Optional information (when bit D1 of ModeAttributes is set)     */

        int  XResolution;         /* horizontal resolution              */
        int  YResolution;         /* vertical resolution                */
        char XCharSize;           /* character cell width               */
        char YCharSize;           /* character cell height              */
        char NumberOfPlanes;      /* number of memory planes            */
        char BitsPerPixel;        /* bits per pixel                     */
        char NumberOfBanks;       /* number of banks                    */
        char MemoryModel;         /* memory model type                  */
        char BankSize;            /* bank size in kb                    */
        char Dummy[246]           /* Info block must be >256 Bytes      */
     } Mode_Info;

char     *Msg_Header[7] = {
     "                                                                  "
     "                                                                  "
     "          VESA BIOS Demostration Program to detect VESA BIOS      "
     "          and to display list of modes supported by the BIOS.     "
     "                 The following modes are supported:               "
     "                                                                  "
     "                                                                  "

/************************************************************************/
/* Main program                                                         */
/************************************************************************/

main()
     {
     void far *farptr;
     int  i, lines;
     union     REGS regs;
     struct    SREGS     sregs;
```

```
/*************************************************************/
/* Force into text mode                                      */
/*************************************************************/

regs.x.ax = 0x03;              /* Setup for mode 3          */
int86(0x10, &regs, &regs);     /* Use BIOS to set mode 3    */
for (i = 0; i < 7; i++)        /* Print header message      */
     printf("n%s",Msg_Header[i]);

/*************************************************************/
/* Check if VESA BIOS is present                             */
/*************************************************************/

regs.x.ax = 0x4F00;              /* VESA BIOS call          */
farptr = (void far *)&VESA_Info;/* Fetch address of buffer */
sregs.es = FP_SEG(farptr);       /* Place address into parm list */
regs.x.di= FP_OFF(farptr);
int86x(0x10, &regs, &regs, &sregs);      /* Try VESA BIOS    */
                     /* Check status and signature          */

if ((regs.x.ax != 0x004F) ||
   VESA_Info.VESASignature[0] != 'V' ||
   VESA_Info.VESASignature[1] != 'E' ||
   VESA_Info.VESASignature[2] != 'S' ||
   VESA_Info.VESASignature[3] != 'A')
     {
     printf("\n...Error: Cannot locate VESA BIOS\n");
     exit(-1);
     }

/*************************************************************/
/* Loop over modes, displaying info for each                 */
/*************************************************************/

for (i = 0; VESA_Info.VideoModePtr[i] != 0xFFFF; i++)
     {

     /* Display mode number                                   */

     printf("\n    %4Xh",VESA_Info.VideoModePtr[i]);
     printf('' (%s):",
          VESA_Info.VideoModePtr[i] & 0x0100 ? "VESA"   "OEM");
     regs.x.ax = 0x4F01;       /* Fn = Return info           */
     regs.x.cx = VESA_Info.VideoModePtr[i];/* Mode           */
     farptr = (void far *)&Mode_Info;/* Fetch addr of buffer  */
     sregs.es = FP_SEG(farptr);     /* Place addr to parm     */
     regs.x.di= FP_OFF(farptr);
     int86x(0x10, &regs, &regs, &sregs);   /* Get info        */
     if (regs.x.ax != 0x004F) /* Check status                 */
         {                  /* and quit if bad               */
       printf("...Error: Cannot get mode info\n");
         exit(-1);
         }

     /* Display mode type (text or graphics)                  */

     printf(" %s ",
         (Mode_Info.ModeAttributes & 0x0010) ? "Graphics":"Text   ");

     /* Display mode resolution                               */

     if (Mode_Info.ModeAttributes & 0x0002)
         {
         printf(" %4d x%4d ",
            Mode_Info.YResolution, Mode_Info.XResolution);

     /* Display number of colors                              */

          if (Mode_Info.ModeAttributes & 0x0008)
              printf("%3d Colors",
                  0x0001 << (Mode_Info.BitsPerPixel));
```

```
          else
              printf("Mono");
          }
     else
          printf("Optional info not available\n");
     if (!((i+8) % 23))  /* Pause if screen full             */
          {
          printf("\nPress <Enter> to continue...'');
          getchar();
     }
    }

/*****************************************************************/
/* Cleanup and exit                                              */
/*****************************************************************/

exit(0);
}
```

# 21

# Displays for SuperVGAs

# Introduction

Unlike earlier IBM display adapters (including EGA) that used a digital (TTL) interface to the display, the VGA requires an analog display. Analog displays are capable of displaying many more colors than digital displays. The 256-color mode of the VGA permits the display of color photographic images with a high degree of realism.

Early display adapters (MDA and CGA) were designed to support either monochrome displays or color displays, but not both. The EGA will support either, but only in specific display modes. Color display modes must be used with a color display, and monochrome display modes must be used with a monochrome display.

The VGA includes no such restrictions. All display modes of the VGA display adapter are available regardless of what display is being used. If a monochrome display is used in a color display mode, the colors will be translated into shades of gray. If a color display is used in a monochrome mode, a monochrome image will be displayed.

IBM markets two VGA-compatible displays; a color display and a monochrome display. These displays include a feature referred to as *automatic monitor detection* or *display detection*. The IBM VGA BIOS will automatically detect what type of display is connected (color or monochrome) and configure itself accordingly. The detection scheme works by reading the voltage level on two pins of the display connector to identify the display type. Not all VGA-compatible displays support this feature, nor do all VGA-compatible adapters support it. Some VGA-compatible adapters still use configuration switches to set the default operating mode.

A large number of VGA-compatible displays are available. Some are purely meant as VGA displays; some, such as the NEC Multisync and Nanao Flexscan, can be switched from digital to analog mode and are useable with either EGA or VGA; some are designed for higher resolutions but include VGA compatibility also.

# Operation of CRT Displays

Cathode Ray Tube (CRT) displays, colors are generated by a beam of electrons which strike the phosphorus coating on the back of the CRT screen and cause it to glow. The electron beam is swept across the display screen from left to right in a series of horizontal lines. At the same time, its intensity is modulated to produce display patterns. The electron beam must continuously redraw the pattern on the screen 50, 60 or 70 times a second, depending on the display used. This process is called **Display Refresh** or **Screen Refresh**.

The sweep pattern of the electron beam on the display screen is called the **Raster**. The beam begins in the upper left corner of the display and sweeps right. When it reaches the right edge of the screen, the beam is shut off (**Horizontal Blanking**) and then rapidly brought back to the left edge of the screen (**Horizontal Retrace**) to begin the next horizontal scan just below the previous one.

After all horizontal scans have been completed, the electron beam will end up in the lower right corner of the screen. At this point the beam is shut off (**Vertical Blanking**) and then rapidly brought back up to the upper left corner (**Vertical Retrace**) so the next raster can begin. This process is represented in Figure 21-1.

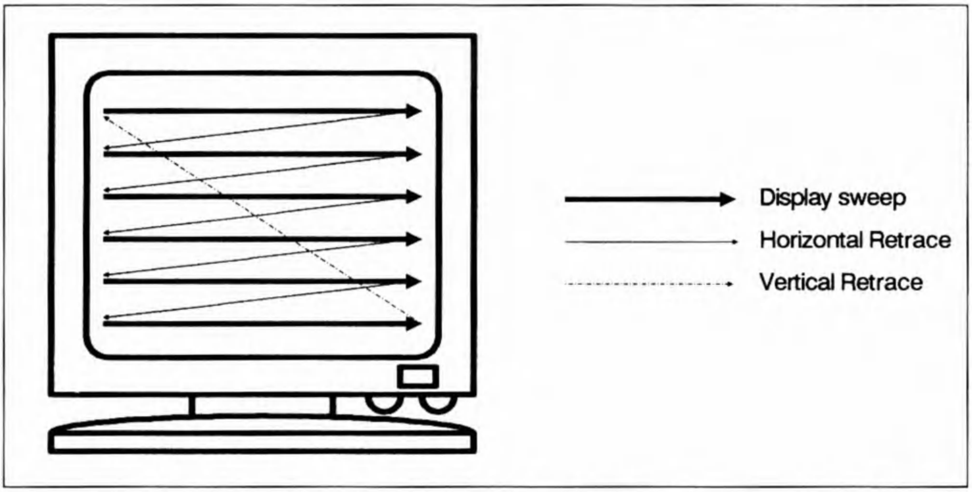


**Figure 21-1. Raster scan**

The entire display pattern can be considered as a long serial string of bits which are fed to the electron beam as it passes over the display screen. The horizontal resolution of the display is equal to the number of bits which can be displayed on one horizontal scan line. The area of the screen which is lighted by a single bit in this data stream is called a **Pixel**. The vertical resolution of the display is determined by the number of horizontal scans that are made.

Circuitry internal to the CRT display generates the electron beam (or beams, in color displays) and drives it across the display screen, but the display adapter must be capable of controlling the motion of the electron beam so it can be synchronized with the data stream. By pulsing the **Horizontal Sync** and **Vertical Sync** signals to the display, the adapter controls the timing of horizontal and vertical retrace cycles.

The CRT Controller in any SuperVGA is used to define duration for the various sections of the raster scan. Each CRT controller contains registers for vertical and horizontal parameters. Each of the two sets of parameters includes the following values:

- Display End - border starts (data no longer fetched from display memory) electron beam turned off
- Blanking Starts - electron beam turned off
- Retrace Start - beam reverses direction

- Retrace End - beam starts new scan (top or left)
- Blanking End - border at the start of the scan (top or left)
- Total - display of next scanline starts

The relation of these values is illustrated in Figure 21-2.

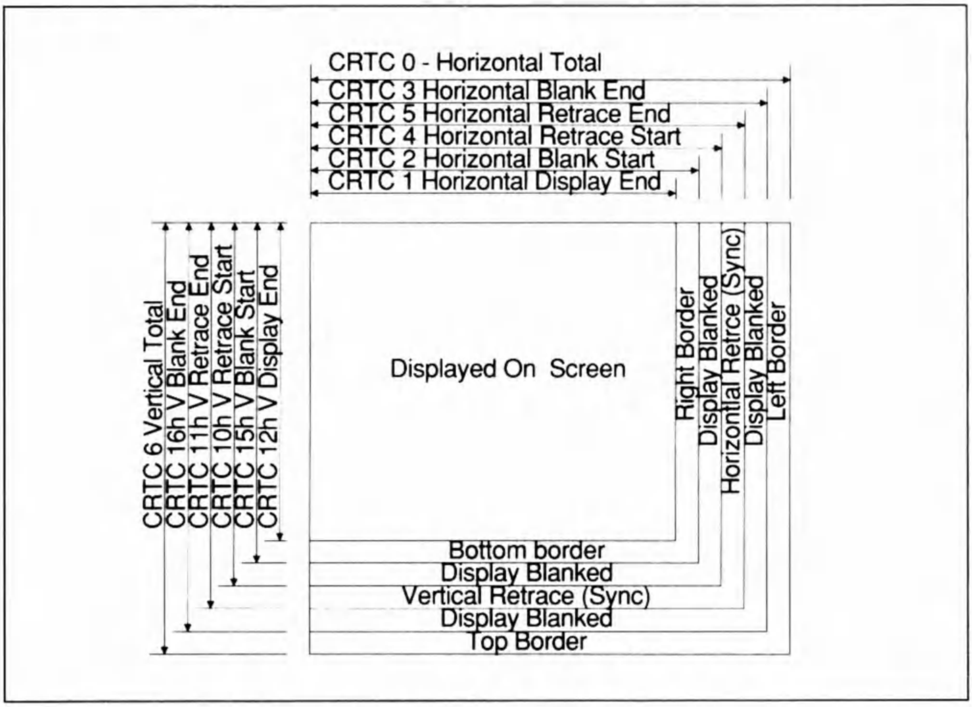


Figure 21-2. CRT Controller timing values

# Factors Affecting Display Resolution

## Scan Frequency vs. Resolution

Displays come in a variety of sizes and capabilities. Display performance is usually defined in three terms: vertical scanning frequency, horizontal scanning frequency and bandwidth. These in turn determine the vertical resolution (total number of scan lines) and horizontal resolution (total number of pixels per scan line) that can be generated.

Vertical scanning frequency determines how many times per second a complete frame is displayed; for SuperVGAs this is typically either 60 or 70 times per second (60 or 70 Hz). Sixty Hz is most common, but higher scanning frequencies are generally perceived as producing less screen flicker and offering a more pleasing display.

Horizontal scanning frequency determines how many scan lines can be displayed in every frame. With a vertical scan rate of 60 Hz, a 640x480 display mode must be able to display at least 28,800 scan lines per second (60 times 480). The actual horizontal scan rate must in fact be slightly higher to allow for the typical 10% overhead needed for retrace times. A horizontal scanning frequency of 31,500 scan lines per second (31.5 kHz) is adequate (this is the specification for the IBM VGA display).

Bandwidth defines how often the electron beam can change intensity, and determines how many pixels can be displayed in each scan line. For 640x480 VGA modes operating at 60 Hz vertical refresh, each scan line takes 1/32,000 of a second. To pack 640 pixels into each scan line, the display must be able to produce 20,800,000 changes each second (assuming no two adjacent pixels are of the same color). The actual bandwidth must in fact be slightly higher to allow for retrace times. VGA uses a 25.125 MHz clock for this mode.

Scan rates for other resolutions can be similarly derived. Since blanking periods and retrace times vary between displays, the computation above is only approximate. Actual specifications for some common displays are shown in Table 21-4 on page 513.

## Shadow Mask and Gun Arrangement

A high bandwidth in the electronics of a display guarantees that the electron gun(s) can be modulated at a high frequency. It does not, however, guarantee that each individual pixel can be clearly distinguished on the display screen. The clarity of pixels on the screen is affected by the construction of the CRT tube itself.

In color displays, three separate electron beams (red, green and blue) are directed through a metal mask (the shadow mask), so that each beam strikes and excites phosphors of a particular color on the back face of the CRT screen (see Figure 21-3 on page 510). Three common types of shadow masks are used: delta, inline, and metal strip.

A delta gun arrangement is the most common and least expensive to manufacture. It is also the one most susceptible to misconvergence (discussed later in this section). An inline shadow mask helps prevent convergence problems. The most accurate (and most expensive) is the single-lens metal-strip arrangement used in the Sony Trinitron (see Figure 21-3). The metal strip also has the advantage of allowing more electrons to strike the phosphor and thus is capable of more brilliant colors.

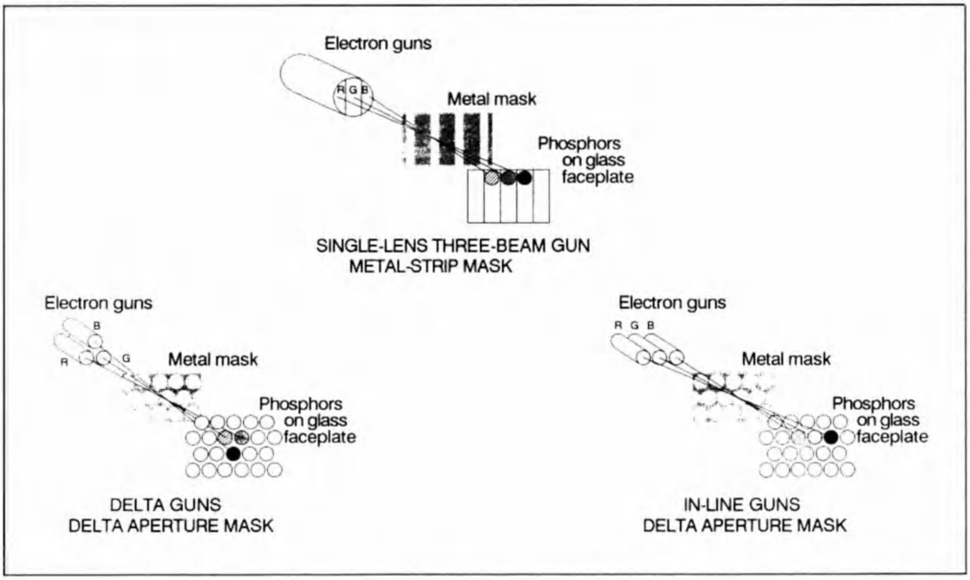


Figure 21-3. Types of shadow masks

Shadow masks are so named because they permit each pixel to be illuminated by the proper gun but shade adjacent pixels of other colors from the energy of the electron beam (creating shadows over the unwanted pixels).

## Dot Pitch and Spot Size

The distance (in microns) between the individual openings in the shadow mask is referred to as the *dot pitch* of the screen. The most common dot pitch for VGA displays is 0.31mm. This pitch determines the theoretical limit on the number of pixels displayable on a screen. Table 21-1 shows vertical and horizontal dimensions for common display sizes. A display with a 16" diagonal screen and 0.31mm dot pitch has a theoretical pixel limit of 912 pixels (283mm / 0.31mm). Many such displays claim 1024 pixel resolution. Note that the metal-strip arrangement does not impose any theoretical limit on the number of horizontal scan lines a screen can support.

To reduce the number of pixels that can be missed or not illuminated sufficiently, the electron beam shines through several openings in the shadow mask for each pixel. The size of the illuminated area is referred to as the spot size (which also varies with the intensity of the beam). For a display adjusted for medium intensity, the spot size, for 0.31mm shadow mask, is typically around 1mm.

Table 21-1. Raster dimensions as function of display size

| Display Size | Horizontal Dimension | Vertical Dimension |
|---|---|---|
| 13″ | 240mm | 180mm |
| 16″ | 283mm | 213mm |
| 19″ | 348mm | 261mm |
| 21″ | 368mm | 276mm |

## Human Eye and Resolution

The theoretical resolution for a display screen often falls below the rated resolution. An explanation of this seeming contradiction lies in the human visual system. It is beyond the scope of this text to go into detail on this subject, but the following factors contribute to the way an image is perceived by the human eye:

- The shadow mask creates distinct points on the screen; pixels are a measurable distance apart from each other.
- Pixels are not illuminated all at once. At any one time only one pixel is being illuminated by the electron beam.
- Pixels change intensity as the electron beam passes over them and as the electron beam changes intensity.
- The eye's response to intensity variations in the discrete pixels is integrated by the visual system into a single image in the brain.

By carefully choosing the pattern to be displayed, results can be quite contrary to rated specifications. Display selection tends to be a subjective process and is dependent on the application for which it will be used.

**Brightness** (Intensity or Luminance) Typically defined in terms of footlamberts (ft-L), brightness is usually specified by a range of values. The dimmest of displays provide brightness levels in the range of six to eight ft-L, while the brightest may range from ten to thirty ft-L. The brightness of the display is determined by phosphor type (short persistence phosphors tend to be brighter), electron beam strength, and by the amount of energy allowed to pass through the shadow mask.

**Blooming** is used to describe the phenomenon where the spot size increases with an increase in intensity.

**Misconvergence** is the alignment error of the red, blue and green guns, as a mean distance between centers of color spot pairs (R-G, R-B and G-R). For a 19-inch display, look for less then 0.5mm of misconvergence at the edges of the screen and less then 0.2mm at the center of the screen.

# Specifications for Common SuperVGA Displays

## Interface Type

VGA adapters require analog RGB displays. EGA adapters require TTL displays. Some displays support both types of interfaces by means of a switch labeled *Analog/TTL*. Some displays can automatically detect the type and switch accordingly.

## Video Connector Type

The IBM standard connector type for VGA displays is a 15 pin D type connector. Older IBM TTL displays used 9-pin D type connectors. These connectors are so named because their shape resembles that of an upper case letter D. Table 21-2 shows the standard pinout for a 15-pin analog video connector.

**Table 21-2. Standard VGA 15-pin video connector pinout**

| | |
|---|---|
| 01 - Red | 09 - Key (missing pin) |
| 02 - Green | 10 - Sync Return (ground) |
| 03 - Blue | 11 - Monitor ID bit 0 |
| 04 - Monitor ID bit 2 | 12 - Monitor ID bit 1 |
| 05 - Not used | 13 - Horizontal Sync (see Table 21-3) |
| 06 - Red Return (ground) | 14 - Vertical Sync (see Table 21-3) |
| 07 - Green Return (ground) | 15 - Not used |
| 08 - Blue Return (ground) | |

**Table 21-3. Standard VGA sync polarity vs scan line count**

| Vertical Sync | Horizontal Sync | Line Count |
|---|---|---|
| + | + | Reserved |
| − | + | 400 lines |
| + | − | 350 lines |
| − | − | 480 lines |

When examining specifications for various monitors, Table 21-4 can be used as a guide to convert between vertical and horizontal refresh rates, and maximum resolutions of the display.

Table 21-4. Typical horizontal and vertical refresh rates

| Mode | Resolution | Vertical Refresh | Horizontal Refresh | Original Board | Display Mnemonic |
|---|---|---|---|---|---|
| | STANDARD MODES | | | | |
| 0, 1 | 320x200 | 60 Hz | 15.75 kHz | CGA | CGA |
| 2, 3 | 640x200 | 60 Hz | 15.75 kHz | CGA | CGA |
| 7 | 720x350 | 50 Hz | 18.43 kHz | MDA | MDA |
| 2*, 3*, | 640x350 | 60 Hz | 21.85 kHz | EGA | EGA |
| F, 10 | 640x350 | 60 Hz | 21.85 kHz | EGA | EGA |
| 2+,3+,7+ | 720x400 | 70 Hz | 31.50 kHz | VGA | VGA |
| F, 10 | 640x350 | 70 Hz | 31.50 kHz | VGA | VGA |
| 11, 12 | 640x480 | 60 Hz | 31.50 kHz | VGA | VGA |
| 13 | 320x200 | 70 Hz | 31.50 kHz | VGA | VGA |
| | ENHANCED GRAPHICS MODES | | | | |
| 6Ah | 800x600 | 56/60 Hz | 35.20/37.88 kHz | SuperVGA | SuperVGA |
| | 1024x768 | 43.48 Hz | 35.52 kHz | 8514/A | 8514 |
| | 1024x768 | 60 Hz | 48.00 kHz | SuperVGA | XL |
| | 1280x1024 | 60 Hz | 64.00 kHz | SuperVGA | XL |
| | ENHANCED TEXT MODES | | | | |
| 132 col | 1056x350 | 70 Hz | 31.50 kHz | SuperVGA | VGA |
| 100 col | 800x480 | 60 Hz | 31.50 kHz | SuperVGA | VGA |

# Selecting a Display for SuperVGA

Monitor selection can be a very subjective process. Each individual responds differently to various displays. Some people are more sensitive to color purity, while others are more sensitive to flicker or brightness. The best display will also depend on the intended application. Displays with crisp text may not always be the best to display complex drawings found in a CAD environment.

Several criteria that one should consider when selecting a display are:

- **Maximum resolution**. Will the display support the high resolution display modes of your VGA board? Common display resolutions include are 640x480, 800x600, and 1024x768. There are several other factors related to resolution, such as brightness, shadow mask type, blooming, spot size, and color purity. These are discussed in the section titled "Factors Affecting Display Resolution" on page 508.

Figure 21-4 illustrates the different classes of displays, and their typical maximum resolutions. Display classes used in Figure 21-4 are the same as those used in Table 21-4.

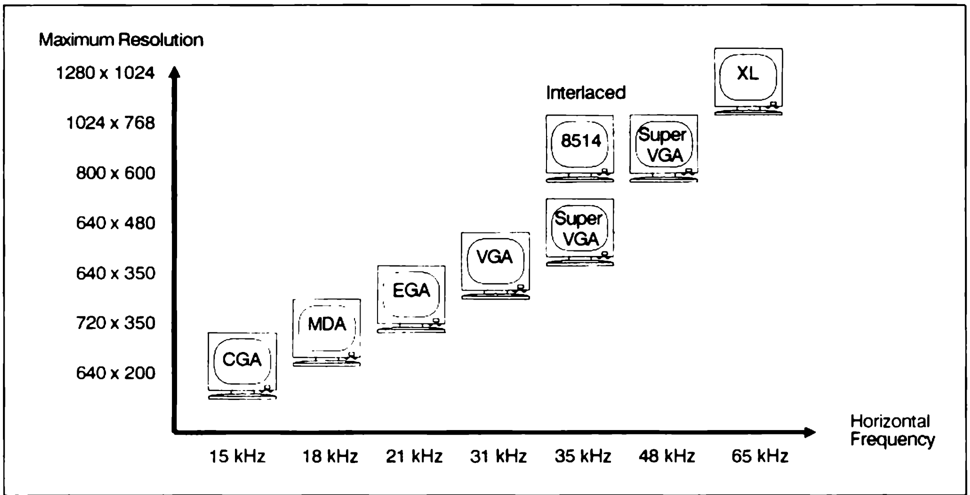


Figure 21-4. Display performance levels

- **Minimum resolution.** Displays that are capable of operating at the highest resolutions may not operate at very low resolutions. Since VGA adapters perform double scanning at the lowest resolutions, this is normally only a concern if the display must also be used with an EGA or CGA adapter.
- **Automatic size adjustment**. Will the display automatically adjust the picture size at different resolutions to optimally fill the visible screen? Some displays which call themselves autosizing will adjust sizes only for VGA resolutions (320x400, 640x400, 720x400, 640x350, and 640x480); at other resolutions, they have to be adjusted manually. Other displays may have to be first programmed: they have to be adjusted once for each frequency or resolution a particular video board supports, and from then on the display will recognize the frequency and adjust as initially programmed.
- **Automatic display detection.** If your VGA board supports automatic display detection, it is important that your display be properly recognized by the board.
- **SuperVGA-scanning capability.** Will it operate at a large number of different resolutions, or only at two or three fixed resolutions? Many of the displays that are now available will operate at virtually any resolution between the minimum and maximum supported by that monitor.
- **VGA compatibility**. To operate properly with a VGA, the display must not only support VGA resolutions, it must also be compatible with the video timing generated by

a VGA adapter. The most significant timing specifications for a display are horizontal scan rate (which may range from about 20 KiloHertz to about 50 KiloHertz) and vertical scan rate (which is typically in the range of 50 Hertz to 70 Hertz). IBM VGA is 31.5 kHz and 60 or 70Hz.

- **Interlaced vs non-interlaced.** Virtually all displays are non-interlaced at resolutions up to 800x600 pixels. At higher resolutions, some of the less expensive displays must operate in interlaced mode. This can produce an annoying flickering of the screen. Figure 21-5 illustrates the basic operation of an interlaced display. Interlaced displays are becoming more common since IBM introduced the 8514 display, which connects to the IBM 8514/A display adapter and operates in interlaced mode at a resolution of 1024 pixels horizontally by 768 pixels vertically.

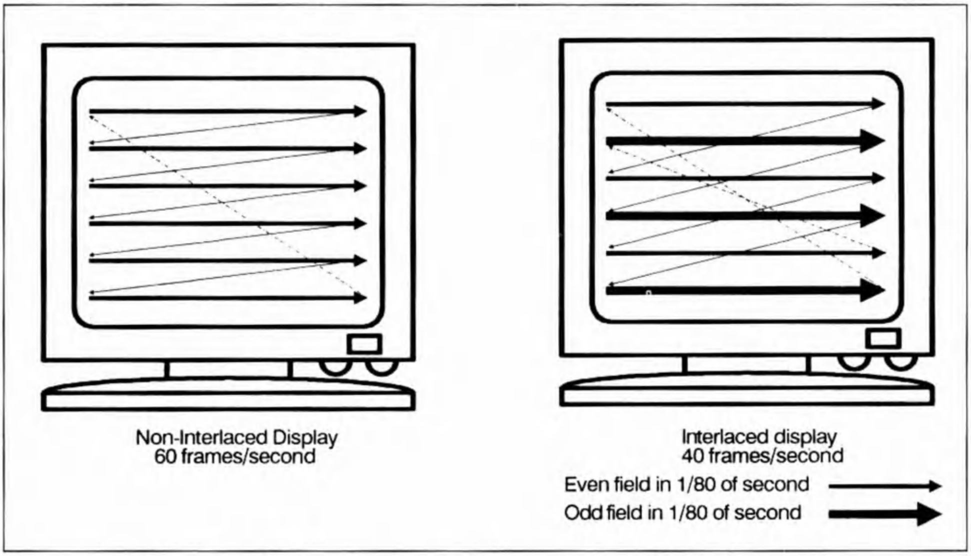


**Figure 21-5. Operation of interlaced displays**

- **Cost**. The least expensive displays (under $200) have the smallest screen sizes (12 or 13 inches diagonally) with the lowest resolutions. As the size and resolution increase, so does the cost. Fourteen-inch displays with a resolution of 640x480 are currently priced around $500, and 800x600 around $700. For 19-inch displays that are capable of 1024x768 resolution, the cost jumps to over $1,500. A high quality 21-inch display, capable of 1280x1024 resolution, today costs over $3,000.
- **Persistence** of the phosphor and image flicker. For a CAD application, where large, complex drawings will be displayed and changes are made slowly, a long persistence phosphor is desirable to minimize flicker. For word processing, a short per-

sistence phosphor will minimize the smearing effect of ghosts on the screen when text is scrolled. Persistence also affects brightness and color purity.

## Popular VGA-Compatible Displays

The table that follows is filled with manufacturer supplied data for popular displays. These specifications typically represent figures at which the display operates at its optimum. In practice most of the displays will perform with satisfactory results at resolutions slightly higher. For example, the Sony multiscan was designed to operate at 640x480 (up to 34kHz), but is commonly used to operate at 800x600 (38kHz) and even at 1024x768 interlaced (36kHz).

Over 30 different display manufacturers currently offer SuperVGA-compatible displays; some manufacturers, such as Sony and Mitsubishi, offer over a dozen different models at various levels of performance and quality. Only the most popular models are included here. Appendix F contains an extensive list of display manufacturers, with addresses and phone numbers, for those who need further information.

This section is primarily intended to give the reader a general feel for the range of performance and prices available. Prices in the table are list; street prices can be as much as 40% lower.

Table 21-5. Typical displays used with SuperVGA

| Model | Size | Pitch | Horiz Freq kHz | Vert Freq Hz | Band-width MHz | Maximum Resolution | List Price | Standards Supported |
|---|---|---|---|---|---|---|---|---|
| Amdek 735 | 14" | .31mm | 15-35 | 50-70 | 40 | 800x600 | $745 | MDA,CGA,EGA,VGA,SuperVGA |
| Goldstar 1440 | 13" | .31mm | 13-35 | 45-85 | 30 | 800x560 | $799 | MDA,CGA,EGA,VGA,SuperVGA |
| IBM 8512 (VGA) | 14" | .41mm | 31.5 | 50-70 | 28 | 640x480 | $699 | VGA |
| IBM 8514 | 16" | .31mm | 31-35 | 43-70 | 28 | 1024x768(i) | $1,550 | VGA,8514 |
| Mitsubishi XC1429C | 14" | .28mm | 31.5 | 50-70 | 25 | 640x470 | $658 | VGA |
| Mitsubishi Diamond Scan 14 | 14" | .31mm | 15-36 | 45-90 | 30 | 800x600 | $889 | MDA,CGA,EGA,VGA,SuperVGA (AUM1381) |
| Mitsubishi Diamond Scan 16 | 16" | .31mm | 30-64 | 50-90 | 100 | 1280x1024 | $2,030 | VGA,SuperVGA,8514,XL (HL6605) |
| Mitsubishi Diamond Scan 20 | 20" | .31mm | 30-64 | 50-90 | 100 | 1280x1024 | $3,445 | VGA,SuperVGA,8514,XL (HL6905) |
| Nanao 9070S | 16" | .31mm | 20-50 | 50-80 | 50 | 1024x768 | $1,700 | EGA,VGA,SuperVGA,8514,XL |
| Nanao 9400 | 19" | .31mm | 30-65 | 55-75 | 120 | 1280x1024 | $3,800 | MDA,CGA,EGA,MDA,XL |
| NEC 2A | 14" | .31mm | 31-35 | 56-70 | 38 | 800x600 | $799 | VGA,SuperVGA |
| NEC XL (discontinued) | 20" | .31mm | 21-50 | 56-80 | 30/65(1) | 1024x768 | $3,200 | EGA,VGA,SuperVGA,XL |
| NEC 3D | 13" | .28mm | 15-38 | 50-90 | 45 | 1024x768(i) | $1,049 | MDA,CGA,EGA,VGA,SuperVGA,8514 |
| NEC 4D | 15" | .28mm | 30-57 | 50-90 | 75 | 1024x768 | $1,799 | VGA,SuperVGA,8514,XL |
| NEC 5D (replaces XL) | 19" | .31mm | 30-66 | 60-90 | 75/110(1) | 1280x1024 | $3,699 | VGA,SuperVGA,8514,XL |
| Princeton Ultrasync | 12" | .28mm | 15-35 | 45-120 | 30 | 800x600 | $849 | MDA,CGA,EGA,VGA,SuperVGA |
| Sampo AlphaScan | 13" | .31mm | 15-37 | 50-70 | 30 | 800x600 | $789 | MDA,CGA,EGA,VGA,SuperVGA |
| Samsung SyncMaster | 14" | .31mm | 15-35 | 58-72 | 30 | 800x560 | $699 | CGA,EGA,VGA,SuperVGA |
| Seiko CM-1440 (was 1430) | 14" | .25mm | 31-40 | 50-90 | 50 | 1024x768(i) | $899 | VGA,SuperVGA,8514 |
| Sony CPD-1302 | 13" | .25mm | 15-34 | 75-100 | 25 | 900x560 | $995 | MDA,CGA,EGA,VGA,SuperVGA,8514 |
| Sony CPD-1304 | 13" | .25mm | 28-50 | 50-100 | 30 | 1024x768 | $1,095 | VGA,SuperVGA,8514,XL |
| Tatung 1595G | 15" | .31mm | 15-37 | 40-120 | 38 | 1024x768(i) | $1,199 | MDA,CGA,EGA,VGA,SuperVGA,8514 |
| Taxan UltraVision 1000 | 20" | .31mm | 30-78 | 50-80 | 200 | 1600x1200 | $3,700 | EGA,VGA,SuperVGA,8514,XL |
| Zenith ZCM-1490 | 14" | .28mm | 31.5 | 50-70 | 28 | 640x480 | $999 | VGA |

(i): Interlaced
(1): Lower bandwidth for DB-9/DB-15 input, higher bandwidth for BNC input

# Appendices

# A

# VGA BIOS Summary

# VGA BIOS Summary

## Function 0 - Mode Select

AH = 0
AL = Mode number (0 to 13H)

## Function 1 - Set Cursor Size

AH = 1
CH = start scan line (0 - 31)
CL = end scan line (0 - 31)

## Function 2 - Set Cursor Position

AH = 2
BH = display page number
DH = Row
DL = Column

## Function 3 - Read Cursor Size and Position

AH = 3
BH = Display page number

**Return Value:**

CH = cursor start scan line
CL = cursor end scan line
DH = cursor row
DL = cursor column

## Function 4 - No Standard Support (Get Light Pen)

## Function 5 - Select Active Page

**Input Parameters:**

AH = 5
AL = display page number

## Function 6 - Scroll Text Window Up (or Blank Window)

AH = 6
AL = number of lines to scroll
(AL = 0 blanks window to all spaces)
BH = text attribute to use when filling blank lines at bottom of window
CH = row number of upper left corner of window
CL = column number of upper left corner of window
DH = row of lower right corner of window
DL = column of lower right corner of window

## Function 7 - Scroll Text Window Down (or Blank Window)

AH = 7
AL = number of lines to scroll
(AL = 0 blanks window to all spaces)
BH = text attribute to use when filling blank lines at top of window
CH = row number of upper left corner of window
CL = column number of upper left corner of window
DH = row of lower right corner of window
DL = column of lower right corner of window

## Function 8 - Read Character and Attribute at Cursor Position

AH = 8
BH = display page number
AL = character code
AH = character attribute (text modes only)

## Function 9 - Write Character and Attribute at Cursor Position

AH = 9
AL = character code
BH = display page number
BL = attribute (text modes) or color value (graphics modes)
CX = repetition count (up to end of current row)

## Function 0Ah - Write Character Only at Cursor Position

AH = 0Ah
AL = character code
BH = display page number
BL = color value (graphics modes)
CX = repetition count (up to end of current row)

## Function 0Bh - Set CGA Color Palette (Modes 4,5,6)

AH = 0Bh

If BH = 0:
BL = graphics background color
 or text border color
If BH = 1:
BL = palette number (0 or 1)

## Function 0Ch - Write Graphics Pixel

AH = 0Ch
AL = pixel value
CX = pixel column number
DX = pixel row number

## Function 0Dh - Read Graphics Pixel

AH = 0Dh
CX = pixel column number
DX = pixel row number

**Return Value:**

AL = pixel value

## Function 0Eh - Write Character and Advance Cursor

AH = 0Eh
AL = character code
BH = page number (text modes only)
BL = character color (graphics modes only)

## Function 0Fh - Get Current Display Mode

AH = 0Fh

**Return Value:**

AH = number of display columns
AL = display mode
BH = active display page

### *Function 10h -Subfunction 0 - Program a Palette Register*

AH = 10h
AL = 00h
BL = palette register number (0 to Fh)
BH = color data (0 to 3Fh)

### *Function 10h -Subfunction 1 - Set Border Color (Overscan)*

AH = 10h
AL = 01h
BH = color data (0 to FFh)

### *Function 10h -Subfunction 2 - Set All Palette Registers*

AH = 10h
AL = 02h
ES:DX = address of 17-byte buffer (16 palette values plus overscan value)

### *Function 10h - Subfunction 3 - Blink/Intensity Attribute Control*

AH = 10h
AL = 03h
BL = 0 - enable background intensify
BL = 1 - enable foreground blink

### Function 10h - Subfunction 7 - Read a Single Palette Register

AH = 10h
AL = 7
BL = register number (0-15)

**Return Value:**

BH = palette register value

### Function 10h - Subfunction 8 - Read Border Color (Overscan) Register

AH = 10h
AL = 8

**Return Value:**

BH = Border Color Register value

### Function 10h - Subfunction 9 - Read All Palette Registers

AH = 10h
AL = 9
ES:DX = address of 17-byte buffer (16 palette values plus overscan value)

**Return Value:**

17 bytes stored at [ES:BX]

### Function 10h - Subfunction 10h - Set a Single DAC Register

AH = 10h
AL = 10h
BX = DAC register number (0 to FFh)
DH = Red intensity level (0 to 3Fh)
CH = Green intensity level (0 to 3Fh)
CL = Blue intensity level (0 to 3Fh)

### Function 10h - Subfunction 12h - Set Block of DAC Registers

AH = 10h
AL = 12h
BX = starting DAC register (0 to 255)
CX = number of registers to set (1 to 256)
ES:DX = address of color table

### Function 10h - Subfunction 13h - Select Color Subset

AH = 10h
AL = 13h

if BL = 0: Select mode
  BH = 0: 4 subsets of 64 colors
  BH = 1: 16 subsets of 16 colors

if BL = 1: Select subset
  BH = subset (0-16)

### Function 10h - Subfunction 15h - Read a Single DAC Register

AH = 10h
AL = 15h
BX = DAC register number (0-255)

**Return Value:**

DH = red intensity level (0 to 3Fh)
CH = green intensity level (0 to 3Fh)
CL = blue intensity level (0 to 3Fh)

### Function 10h - Subfunction 17h - Read Block of DAC Registers

AH = 10h
AL = 17h
BX = starting DAC register number (0-255)
CX = number of registers (1-256)
ES:DX = destination address for register data

**Return Value:**

Register data at destination address (3 bytes per register)

### Function 10h - Subfunction 1Ah - Read Subset Status

AH = 10h
AL = 1Ah

**Return Value:**

BH = number of current color subset
BL = 0 if 4 subsets are available
BL = 1 if 16 subsets are available

### Function 10h - Subfunction 1Bh - Convert DAC Registers to Gray Scale

AH = 10h
AL = 1bh
BX = starting DAC register number (0-255)
CX = number of registers (1-256)

### Function 11h - Subfunction 0 - Load Custom Character Generator

AH = 11h
AL = 0
ES:BP = address of character data in system RAM
CX = number of characters to load (1 to 256)
DX = character offset into character generator table
(0 to 255 - for loading a partial character set)
BL = which character generator to load
BH = number of bytes per character (1 to 32)

### Function 11h - Subfunction 1 - Load 8 x 14 Character Set

AH = 11h
AL = 1
BL = which character generator to load (0 to 7)

### Function 11h - Subfunction 2 - Load 8 x 8 Character Set

AH = 11h
AL = 2
BL = which character generator to load (0 to 7)

### Function 11h - Subfunction 3 - Select Active Character Set(s)

AH = 11h
AL = 3
BL(D0,D1,D4) - Selects which character generator will be active for a character with attribute bit 3 = 0
BL(D2,D3,D5) - Selects which character generator will be active for a character with attribute bit 3 = 1

### Function 11h - Subfunction 4 - Load 8 x 16 Character Set

AH = 11h
AL = 4
BL = which character generator to load (0-7)

### Function 11h - Subfunctions 10h, 11h, 12h, 14h

Function 11h - Subfunctions 10h, 11h, 12h and 14h are identical to functions 0, 1, 2 and 4, except that CRTC is reprogrammed to match the selected character size.

### Function 11h - Subfunction 20h - Initialize INT 1Fh Vector (Modes 4-6)

AH = 11h
AL = 20h
ES:BP = Pointer to character definitions

### Function 11h - Subfunction 21h - Set Graphics Mode to Display Custom Character Set

AH = 11h
AL = 21h
ES:BP = address of custom character table in system RAM
CX = bytes per character
BL = number of character rows to be displayed:
1 = 14 character rows
2 = 25 character rows
3 = 43 character rows
0 = DL contains number of character rows

### Function 11h - Subfunction 22h - Set Graphics to Display 8 x 14 Text

AH = 11h
AL = 22H
BL indicates number of character rows on screen
1 = 14 character rows
2 = 25 character rows
3 = 43 character rows
0 = DL contains number of character rows
Not all values will result in satisfactory appearance

### Function 11h - Subfunction 23h - Initialize Graphics Mode to Display 8 x 8 Text

AH = 11h
AL = 23H
BL indicates number of character rows on screen
1 = 14 character rows
2 = 25 character rows
3 = 43 character rows
0 = DL contains number of character rows
Not all values will result in satisfactory appearance

### Function 11h - Subfunction 24h - Initialize Graphics Mode to Display 8 x 16 Text

AH = 11h
AL = 24H
BL indicates number of character rows on screen
BL = 1 - 14 character rows
BL = 2 - 25 character rows
BL = 3 - 43 character rows

### Function 11h - Subfunction 30h - Return Information About Current Character Set

AH = 11h
AL = 30h
BH = Information type requested
BH = 0: return current INT 1FH pointer
BH = 1: return current INT 43H pointer
BH = 2: return pointer to Enhanced (8x14) character set
BH = 3: return pointer to CGA (8x8) character set
BH = 4: return pointer to upper half of CGA 8x8 char set
BH = 5: return pointer to alternate 9x14 monochrome characters
BH = 6: return pointer to 8x16 characters
BH = 7: return pointer to alternate 9x16 characters

**Return Values:**

CL = character height (number of rows in a character)
DL = character rows on screen - 1
ES:BP = return pointer

### Function 12h - Subfunction 10h - Return VGA Information

AH = 12h
BL = 10h

**Return Values:**

BH = 0 Color mode in effect (3Dx)
1 Mono mode in effect (3Bx)
BL = Memory size: 0 = 64k, 1 = 128k, 2 = 192k, 3 = 256k
CH = Feature bits
CL = EGA switch settings

### Function 12h - Subfunction 20h - Revector Print Screen (INT 05h) Interrupt

AH = 12h
BL = 20h

### Function 12h - Subfunction 30h - Select Scan Line Count for Next Text Mode

AH = 12h
AL = Number of scan lines: 0 = 200, 1 = 350, 2 = 400
Will take effect on next mode select for modes 0 to 3 and 7.
BL = 30h

**Return Values:**

AL = 12h indicating that function is supported (0 if VGA not active)

### Function 12h - Subfunction 31h - Enable/Disable Palette Load During Mode Set

AH = 12h
AL = 0 enable (default), 1 disable
BL = 31h

**Return Values:**

AL = 12h indicating that function is supported (0 if VGA not active)

## Function 12h - Subfunction 32h - Enable/Disable VGA Access

AH = 12h
AL = 0 enable, 1 disable I/O and memory access to VGA
BL = 32h

**Return Values:**

AL = 12h indicating that function was performed (AL was 0 or 1)

## Function 12h - Subfunction 33h - Enable/Disable Gray Scale Summing

AH = 12h
AL = 0 enable, 1 disable gray scale summing
BL = 33h

**Return Values:**

AL = 12h indicating that function is supported (0 if VGA not active)

## Function 12h - Subfunction 34h - Enable/Disable CGA/MDA Cursor Emulation.

AH = 12h
AL = 0 enable, 1 disable CGA cursor emulation
BL = 34h

**Return Values:**

AL = 12h indicating that function is supported (AL was 0 or 1)

## Function 12h - Sub-function 35h - Switch Displays.

AH = 12h
AL = Select video
0 - Initial adapter video system off (before call with AL = 1)
1 - Initial motherboard video system on (after call with AL = 0)
2 - Switch to inactive BIOS and video system (before call with AL = 3)
3 - Initialize video system with parameters in ES:DX (after call with AL = 0 or 2)
BL = 35h
ES:DX = address of 128-byte save area (for AL = 0, 2, or 3)

**Return Values:**

AL = 12h indicating that function is supported (0 if VGA not active)

### Function 12h - Subfunction 36h - Display On/Off

AH = 12h
AL = 0 enable, 1 disable video output (maximum access to display memory)
BL = 36h

**Return Values:**

AL = 12h indicating that function is supported (0 if VGA not active)

## Function 13h - Write Text String

AH = 13h
BH = display page number
CX = character count (length of string)
DH = row for start of string
DL = column for start of string
ES:BP = address of source text string in system RAM
AL = mode: 0: BL = Attribute for all characters - Cursor is not updated
1: BL = Attribute for all characters - Cursor is updated
2: String contains alternating character codes and Attributes - Cursor is not updated
3: String contains alternating character codes and Attributes - Cursor is updated

### Function 1Ah - Subfunction 0 - Read Display Configuration Code

AH = 1Ah
AL = 0

**Return Values:**

AL = 1Ah
BL = primary display
BH = secondary display
Display information is interpreted as follows:
0 = no display
1 = MDA
2 = CGA
3 = EGA with ECD display
4 = EGA with CD display
5 = EGA with Monochrome Display
6 = PGC (Professional Graphics Controller)
7 = VGA with monochrome display
8 = VGA with color display

0Bh = MCGA with monochrome display
0Ch = MCGA with color display

### Function 1Ah - Subfunction 1 - Write Display Configuration Code

AH = 1Ah
AL = 1
BL = primary display info
BH = secondary display info
For an explanation of info codes, see sub-function 0.

**Return Value:**

AL = 1Ah

## Function 1Bh - Return VGA Status Information

AH = 1Bh
BX = 0
ES:DI = pointer to 64 byte buffer for return data

**Return Values:**

AL = 1Bh
The return buffer will contain information as shown in table A-1

**Table A-1. VGA Functionality and video state information**

| Byte Number | Size | Contents |
|---|---|---|
| 0 | dword | Pointer to STATIC FUNCTIONALITY TABLE (see table A-2) |
| 4 | byte | Current display mode |
| 5 | word | Number of character columns |
| 7 | word | Size of video data area (REGEN BUFFER) in bytes |
| 9h | word | Current offset within REGEN BUFFER |
| 0Bh | 8 words | Cursor positions, two words per page, for up to 8 pages |
| 1Bh | byte | Cursor end |
| 1Ch | byte | Cursor start |
| 1Dh | byte | Current display page |
| 1Eh | word | CRT Controller address (3B4h or 3D4h) |
| 20h | byte | CGA/MDA mode register value (value of 3B8h/3D8h) |
| 21h | byte | CGA/MDA color register value (value of 3B9h/3D9h) |
| 22h | byte | Number of text rows |
| 23h | byte | Character height (in scan lines) |
| 25h | byte | Display Configuration Code (active display) |

**Table A-1. VGA Functionality and video state information** *(continued)*

| Byte Number | Size | Contents |
|---|---|---|
| 26h | byte | Display Configuration Code (inactive display) |
| 27h | word | Number of colors in current mode (0 for mono modes) |
| 29h | byte | Number of display pages in current mode |
| 2Ah | byte | Number of scan lines in current mode:<br>0 = 200, 1 = 350, 2 = 400, 3 = 480 |
| 2Bh | byte | Primary character generator (0-7) |
| 2Ch | byte | Secondary character generator (0-7) |
| 2Dh | byte | Miscellaneous state information:<br>D5 = 1 - Blinking enabled<br>D5 = 0 - Background intensify enabled<br>D4 = 1 - CGA cursor emulation enabled<br>D3 = 1 - Default palette initialization disabled<br>D2 = 1 - Monochrome display attached<br>D1 = 1 - Gray scale conversion enabled<br>D0 = 1 - All modes supported on all monitors |
| 2Eh | byte | Reserved |
| 2Fh | byte | Reserved |
| 30h | byte | Reserved |
| 31h | byte | Size of display memory: 0 = 64KB 1 = 128KB 2 = 192KB 3 = 256KB |
| 32h | byte | Save Pointer State Information<br>D5 = 1 - DCC extension is active (DCC override)<br>D4 = 1 - Palette override active<br>D3 = 1 - Graphics font override active<br>D2 = 1 - Alpha font override active<br>D1 = 1 - Dynamic save area active<br>D0 = 1 - 512 Character set active |
| 33h to 33F | | Reserved |

**Table A-2. VGA Static functionality table**

| Byte Number | Size | Contents |
|---|---|---|
| 0 | byte | Video modes supported (1 indicates mode supported)<br>D7 - mode 7<br>D6 - mode 6<br>D5 - mode 5<br>D4 - mode 4<br>D3 - mode 3<br>D2 - mode 2<br>D1 - mode 1<br>D0 - mode 0 |

**Table A-2. VGA Static functionality table** *(continued)*

| Byte Number | Size | Contents |
|---|---|---|
| 1 | byte | Video modes supported (1 indicates mode supported) |
| | | D7 - mode 0Fh |
| | | D6 - mode 0Eh |
| | | D5 - mode 0Dh |
| | | D4 - mode 0Ch |
| | | D3 - mode 0Bh |
| | | D2 - mode 0Ah |
| | | D1 - mode 9 |
| | | D0 - mode 8 |
| 2 | byte | Video modes supported (1 indicates mode supported) |
| | | D7 - Reserved |
| | | D6 - Reserved |
| | | D5 - Reserved |
| | | D4 - Reserved |
| | | D3 - mode 13h |
| | | D2 - mode 12h |
| | | D1 - mode 11h |
| | | D0 - mode 10h |
| 3 to 6 | | Reserved |
| 7 | byte | Scan line available in text modes (1 indicates supported) |
| | | D2 - 400 lines |
| | | D1 - 350 lines |
| | | D0 - 200 lines |
| 8 | byte | Maximum number of simultaneously displayable character generators |
| 9 | byte | Number of available character generators |
| 0Ah | byte | Miscellaneous BIOS capabilities (1 indicates function supported) |
| | | D7 - Color paging (fn 10h) |
| | | D6 - DAC loading (fn 10h) |
| | | D5 - EGA palette loading (fn 10h) |
| | | D4 - CGA cursor emulation (fn 1) |
| | | D3 - Palette loading after mode set (fn 0) |
| | | D2 - Character generator loading (fn 11h) |
| | | D1 - Gray scale summing (fn 10h and 12h) |
| | | D0 - All modes on all displays |
| 0Bh | byte | Miscellaneous BIOS capabilities (1 indicates function supported) |
| | | D7 - Reserved |
| | | D6 - Reserved |
| | | D5 - Reserved |
| | | D4 - Reserved |
| | | D3 - DCC (fn 1Ah) |

Table A-2. VGA Static functionality table *(continued)*

| Byte Number | Size | Contents |
|---|---|---|
| | | D2 - Blink/Intensify select (fn 10h) |
| | | D1 - Save/Restore video state (fn 1Ch) |
| | | D0 - Light pen (fn 4) |
| 0Ch to 0Dh | | Reserved |
| 0Eh | byte | Save area function support (1 indicates supported) |
| | | D7 - Reserved |
| | | D6 - Reserved |
| | | D5 - DCC extensions |
| | | D4 - Palette override |
| | | D3 - Text character generator override |
| | | D2 - Graphics character generator override |
| | | D1 - Dynamic save area |
| | | D0 - 512 simultaneous characters |
| 0Fh | | Reserved |

## Function 1Ch - Subfunction 0 - Return Required Buffer Size

AH = 1Ch
AL = 0
CX = Type of data to be saved
D0 - Registers
D1 - BIOS data area
D2 - DAC registers

**Return Value:**

AL = 1Ch
BX = Required buffer size (in 64 byte blocks)

## Function 1Ch - Subfunction 1 - Save Display Adapter State

AH = 1Ch
AL = 1
CX = Type of data to be saved
D0 - Registers
D1 - BIOS data area
D2 - DAC registers

ES:BX = Pointer to save buffer

**Return Value:**

AL = 1Ch

### Function 1Ch - Subfunction 2 - Restore Display Adapter State

AH = 1Ch
AL = 2
CX = Type of data to be restored
  D0 - Registers
  D1 - BIOS data area
  D2 - DAC registers
ES:BX = Pointer to save buffer

**Return Value:**

AL = 1Ch

# The BIOS Data Area

The BIOS Data Area is a section of the low memory where various BIOS services keep their working variables. Variables used by Video Services are summarized in table A-3. Programs which directly alter the status of the display without using the BIOS calls (such as cursor position in CRTC registers) should update these variables to avoid confusing the BIOS.

**Table A-3. BIOS Data Area**

| Address | Size | Contents | |
|---|---|---|---|
| 0000:0410h | byte | EQUIPMENT_FLAG<br>Bits D4 and D5 of this byte identify the current primary display device:<br>D5 D4 Adapter<br>0 0 VGA<br>0 1 CGA 40x25<br>1 0 CGA 80x25<br>1 1 MDA | |
| 0000:0449h | byte | VIDEO_MODE | (current mode) |
| 0000:044Ah | word | COLUMNS | (number of text columns) |
| 0000:044Ch | word | PAGE_LENGTH | (length of each page in bytes) |
| 0000:044Eh | word | START_ADDR | (Start Address Register value) |

Table A-3. BIOS Data Area *(continued)*

| Address | Size | Contents | |
|---|---|---|---|
| 0000:0450h | 8 words | CURSOR_POSITION | (cursor positions for all pages) |
| 0000:0460h | word | CURSOR_SHAPE | (Cursor Start and End Registers) |
| 0000:0462h | byte | ACTIVE_PAGE | (current active page number) |
| 0000:0463h | word | CRTC_ADDRESS | (3B4h or 3D4h) |
| 0000:0465h | byte | MODE_REG_DATA | (CGA Mode Register setting) |
| 0000:0466h | byte | PALETTE | (CGA Color Register setting) |
| 0000:0484h | byte | ROWS | (number of text rows - 1) |
| 0000:0485h | word | CHAR_HEIGHT | (bytes per char) |
| 0000:0487h | byte | EGA_INFO_1 | |
| | | D7 = bit D7 from AL on most recent mode select.<br>(a one indicates memory was not cleared by mode select)<br>D6,D5 = Display memory size (00 = 64K, 01 = 128K, 10 = 192K, 11 = 256K)<br>D4 = reserved<br>D3 = A zero indicates VGA is the primary display<br>D2 = A one will force the BIOS to wait for Vertical Retrace before writing to display memory.<br>D1 - A one indicates that VGA is in monochrome mode.<br>D0 - A zero means that CGA cursor emulation is enabled. | |
| 0000:0488h | byte | EGA_INFO_2 | |
| | | D4-D7 = Feature connector settings<br>D0-D3 = Switch settings | |
| 0000:0489h | byte | MISC_FLAGS | D7&D4 = Scanline count:<br>0 0 = 350 lines<br>0 1 = 400 lines<br>1 0 = 200 lines<br>1 1 = reserved<br>D6 = Display switching enabled<br>D3 = Default palette loading disabled<br>D2 = Monochrome monitor<br>D1 = Gray scale summing enabled<br>D0 = VGA active |
| 0000:048Ah | byte | DCC_INDEX | Index of current video combination |
| 0000:04A8h | dword | SAVE_AREA_PTR | Pointer to save area (see table A-4) |

**Table A-4. VGA BIOS Save area**

| Byte Number | Size | Contents |
|---|---|---|
| 0 | dword | Mandatory pointer to Video Parameter Table (see table A-5) |
| 4 | dword | Optional pointer to Dynamic Save Area. (This 256 byte table contains 16 palette register values and overscan register value) |
| 8 | dword | Optional pointer to Text Mode Auxiliary Character Set (see table A-6) |
| 0Ch | dword | Optional pointer to Graphics Mode Auxiliary Character Set (see table A-7) |
| 10h | dword | Optional pointer to Secondary Save Area (see table A-8) |
| 14h | dword | Reserved |
| 18h | dword | Reserved |

Note: At system initialization, the Environment Pointer is set to point to an Environment Table in ROM. This default Environment Table has only one entry (the Video Parameter Table Pointer.) To modify the Environment Table, first copy it from ROM to RAM and then update the Environment Pointer.

**Table A-5. VGA BIOS Video Parameter Table**

| Byte Number | Size | Contents |
|---|---|---|
| 0 | | Number of text columns |
| 1 | | Number of text rows |
| 2 | | Character height (in pixels) |
| 3 and 4 | | Display page length (in bytes) |
| | | Sequencer register values: |
| 5 | | Clock Mode Register |
| 6 | | Color Plane Write Enable Register |
| 7 | | Character Generator Select Register |
| 8 | | Memory Mode Register |
| 9 | | Miscellaneous Register |
| | | CRT Controller register values: |
| 0ah | | Horizontal Total Register |
| 0bh | | Horizontal Display End Register |
| 0ch | | Start Horizontal Blanking Register |
| 0dh | | End Horizontal Blanking Register |
| 0eh | | Start Horizontal Retrace Register |
| 0fh | | End Horizontal Retrace Register |
| 10h | | Vertical Total Register |
| 11h | | Overflow Register |
| 12h | | Preset Row Scan Register |
| 13h | | Maximum Scan Line Register |

**Table A-5.** **VGA BIOS Video Parameter Table** *(continued)*

| Byte Number | Size | Contents |
|---|---|---|
| 14h | | Cursor Start |
| 15h | | Cursor End |
| 16h-19h | | Unused |
| 1ah | | Vertical Retrace Start Register |
| 1bh | | Vertical Retrace End Register |
| 1ch | | Vertical Display End Register |
| 1dh | | Offset Register |
| 1eh | | Underline Location Register |
| 1fh | | Start Vertical Blanking Register |
| 20h | | End Vertical Blanking Register |
| 21h | | Mode Control Register |
| 22h | | Line Compare Register |
| | | Attribute Controller Register Values: |
| 23h | | Palette Register 0 |
| 24h | | Palette Register 1 |
| 25h | | Palette Register 2 |
| 26h | | Palette Register 3 |
| 27h | | Palette Register 4 |
| 28h | | Palette Register 5 |
| 29h | | Palette Register 6 |
| 2ah | | Palette Register 7 |
| 2bh | | Palette Register 8 |
| 2ch | | Palette Register 9 |
| 2dh | | Palette Register 10 |
| 2eh | | Palette Register 11 |
| 2fh | | Palette Register 12 |
| 30h | | Palette Register 13 |
| 31h | | Palette Register 14 |
| 32h | | Palette Register 15 |
| 33h | | Mode Control Register |
| 34h | | Screen Border Color (Overscan) Register |
| 35h | | Color Plane Enable Register |
| 36h | | Horizontal Panning Register |
| | | Graphics Controller register values: |
| 37h | | Set/Reset Register |
| 38h | | Set/Reset Enable Register |
| 39h | | Color Compare Register |
| 3ah | | Data Rotate & Function Select Register |
| 3bh | | Read Plane Select Register |
| 3ch | | Mode Register |
| 3dh | | Miscellaneous Register |

**Table A-5. VGA BIOS Video Parameter Table** *(continued)*

| Byte Number | Size | Contents |
|---|---|---|
| 3eh | | Color Don't Care Register |
| 3fh | | Bit Mask Register |

Modes are ordered in the parameter table as follows:

| Table | Mode |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 2 |
| 3 | 3 |
| 4 | 4 |
| 5 | 5 |
| 6 | 6 |
| 7 | 7 |
| 8 | 8 |
| 9 | 9 |
| 10 | A |
| 11 | B |
| 12 | C |
| 13 | D |
| 14 | E |
| 15 | F (64 KByte display RAM) |
| 16 | 10 (64 KByte display RAM) |
| 17 | F (more than 64 KBytes) |
| 18 | 10 (more than 64 KBytes) |
| 19 | 0* |
| 20 | 1* |
| 21 | 2* |
| 22 | 3* |
| 23 | 0+,1+ |
| 24 | 2+,3+ |
| 25 | 7+ |
| 26 | 11 |
| 27 | 12 |
| 28 | 13 |

## Table A-6. VGA BIOS Text Mode Auxiliary Character Set Table

| Byte Number | Size | Contents |
|---|---|---|
| 0 | byte | Bytes per character |
| 1 | byte | Character Map # (0-3 for EGA, 0-7 for VGA) |
| 2,3 | word | # of characters |
| 4,5 | word | first character # |
| 6,7,8,9 | dword | pointer to character set in system memory |
| 10 | byte | character height (in pixels) |
| 11-n | bytes | list of modes this character set is compatible with, terminated by FFh |

## Table A-7. VGA BIOS Graphics Mode Auxiliary Character Set Table

| Byte Number | Size | Contents |
|---|---|---|
| 0 | byte | Number of character rows on display |
| 1,2 | word | Bytes per character |
| 3,4,5,6 | dword | Pointer to character set in system memory |
| 7-n | bytes | List of modes this character set is compatible with, terminated by FFh |

## Table A-8. VGA BIOS Secondary Save Area Table

| Byte Number | Size | Contents |
|---|---|---|
| 0 | word | Length of this table |
| 2 | dword | Pointer to DCC table (see table A-9) |
| 6 | dword | Pointer to second Text Mode Auxiliary Character Set (see table A-6) |
| 0Ah | dword | Pointer to User Palette Table (see table A-10) |
| 0Eh | dword | Reserved |
| 12h | dword | Reserved |
| 16h | dword | Reserved |

## Table A-9. VGA BIOS Device Combination Code Table

| Byte Number | Size | Contents |
|---|---|---|
| 0 | byte | Number of entries in this table |
| 1 | byte | Version number |
| 2 | byte | Maximum display type code |
| 3 | byte | Reserved |

**Table A-9. VGA BIOS Device Combination Code Table** *(continued)*

| Byte Number | Size | Contents |
|---|---|---|
| 4 - n | words | List of valid video combinations, one pair per combination. Pairs are built from the following values:<br>0 = no display<br>1 = MDA<br>2 = CGA<br>3 = EGA with ECD display<br>4 = EGA with CD display<br>5 = EGA with Monochrome Display<br>6 = PGC (Professional Graphics Controller)<br>7 = VGA with monochrome display<br>8 = VGA with color display<br>0Bh = MCGA with monochrome display<br>0Ch = MCGA with color display |

**Table A-10. VGA BIOS User Palette Table**

| Byte Number | Size | Contents |
|---|---|---|
| 0 | byte | Underlining flag: -1 = Off, 0 = Ignore, 1 = On |
| 1 | byte | Reserved |
| 2 | word | Reserved |
| 4 | word | Number of palette registers in the table |
| 6 | word | First palette registers in the table |
| 8 | dword | Pointer to palette register values |
| 0Ch | word | Number of DAC registers in the table |
| 0Eh | word | First DAC register in the table |
| 10h | dword | Pointer to DAC register values (table has 3 bytes per RGB register) |
| 14h | bytes | List of video modes terminated by 0FFh |

# B

# VGA Register Summary

# VGA Register Summary

## Miscellaneous Output Register (I/O Address Write 3C2h, Read 3CCh)

D7 - Vertical Sync Polarity
D6 - Horizontal Sync Polarity
D7 D6
0 0 invalid
0 1 350 lines
1 0 400 lines
1 1 480 lines
D5 - Odd/Even Page Bit
D4 - Disable Video
D3 - Clock Select 1
D2 - Clock Select 0
D1 - Enable/Disable Display RAM
D0 - I/O address select (3Bx vs 3Dx)

## Input Status Register 0 (I/O Address 3C2, Read only)

D7 - Vertical Retrace Interrupt Pending
D6 - Feature connector bit 1
D5 - Feature connector bit 0
D4 - Switch sense bit
D0 to D3 - Unused

## Input Status Register 1 (I/O Address 3BAh/3DAh, Read only)

D7 - unused
D6 - unused
D5 - Diagnostic
D4 - Diagnostic
D3 - Vertical Retrace
D2 - unused
D1 - unused
D0 - Display Enable

## VGA Enable Register (I/O Address 3C3h/46E8h)

D7-D1 - Reserved
D0 - VGA Enable/Disable

## The CRT Controller Registers - 3D4h/3B4, 3D5h/3B5h

Index 0 - Horizontal Total
Index 1 - Horizontal Display Enable
Index 2 - Start Horizontal Blanking
Index 3 - End Horizontal Blanking
  D7 - Test
  D6 - Skew control
  D5 - Skew control
  D0 to D4 - End blanking
Index 4 - Start Horizontal Retrace
Index 5 - End Horizontal Retrace
  D7 - End horizontal blanking bit 5
  D6 - Horizontal retrace delay
  D5 - Horizontal retrace delay
  D0 to D4 - End horizontal retrace
Index 6 - Vertical Total
Index 7 - Overflow Register
  D7 - Vertical Retrace Start (Bit 9)
  D6 - Vertical Display Enable End (Bit 9)
  D5 - Vertical Total (Bit 9)
  D4 - Line Compare (Bit 8)
  D3 - Start Vertical Blank (Bit 8)
  D2 - Vertical Retrace Start (Bit 8)
  D1 - Vertical Display Enable End (Bit 8)
  D0 - Vertical Total (Bit 8)
Index 8 - Preset Row Scan
  D7 - Unused
  D6 - Byte panning control
  D5 - Byte panning control
  D0 to D4 - Preset Row Scan
Index 9 - Maximum Scan Line/Character Height
  D7 - Double Scan
  D6 - Bit D9 of Line Compare register
  D5 - Bit D9 of Start Vertical Blank register
  D4-D0 - Maximum Scan Line

## The CRT Controller Registers - 3D4h/3B4, 3D5h/3B5h *(continued)*

Index 0Ah - Cursor Start
- D7,D6 - reserved (0)
- D5 - Cursor Off
- D4-D0 - Cursor Start

Index 0Bh - Cursor End
- D7 - reserved
- D6,D5 - Cursor Skew
- D4-D0 - Cursor End

Index 0Ch - Start Address (High Byte)
Index 0Dh - Start Address (Low Byte)
Index 0Eh - Cursor Location (High Byte)
Index 0F - Cursor Location (Low Byte)
Index 10h - Vertical Retrace Start
Index 11h - Vertical Retrace End
- D7 - Write protect CRTC registers 0 to 7
- D6 - Refresh cycle select
- D5 - Enable vertical interrupt (when 0)
- D4 - Clear vertical interrupt (when 0)
- D0 to D3 - Vertical retrace end

Index 12h - Vertical Display Enable End
Index 13h - Offset/Logical Screen Width
Index 14h - Underline Location Register
- D7 - Reserved
- D6 - Double word mode
- D5 - Count by 4
- D0 to D4 - Underline Location

Index 15h - Start Vertical Blanking
Index 16h - End Vertical Blanking
Index 17h - Mode Control Register
- D7 - Enable vertical and horizontal retrace
- D6 - Byte mode (1), Word mode (0)
- D5 - Address wrap
- D4 - Reserved
- D3 - Count by two
- D2 - Multiply vertical by 2 (use half in CRTC 8,10h,12h,15h,18h)
- D1 - Select row scan counter (allows 400 line modes)
- D0 - Compatibility mode support (enable interleave)

Index 18h - Line Compare Register

## Sequencer Registers - 3C4h, 3C5h

Index 0 - Reset Register
  D7-D2 - reserved
  D1 - Synchronous Reset
  D0 - Asynchronous Reset
Index 1 - Clock Mode Register
  D7,D6 - Reserved
  D5 - Display Off
  D4 - Allow 32 bit fetch (not used in standard modes)
  D3 - Divide dot clock by 2 (used in some 320x200 modes)
  D2 - Allow 16 bit fetch (used in mono graphics modes)
  D1 - Reserved
  D0 - Enable (0) 9 dot characters (mono text and 400 line text modes)
Index 2 - Color Plane Write Enable Register
  D7,D6 - reserved
  D3 - plane 3 write enable
  D2 - plane 2 write enable
  D1 - plane 1 write enable
  D0 - plane 0 write enable
Index 3 - Character Generator Select Register
  D7,D6 - reserved
  D5 - Character generator table select A (MSB)
  D4 - Character generator table select B (MSB)
  D3,D2 - Character generator table select A
  D1,D0 - Character generator table select B
Index 4 - Memory Mode Register
  D4 to D7 - Reserved
  D3 - Chain 4 (address bits 0&1 to select plane, mode 13h)
  D2 - Odd/Even (address bit 0 to select plane 0&2 or 1&3, text modes)
  D1 - Extended memory (disable 64k modes)
  D0 - Reserved

## Graphics Controller Registers - 3CEh, 3CFh

Index 0 - Set/Reset Register
  D7-D4 - reserved (0)
  D3 - fill data for plane 3
  D2 - fill data for plane 2
  D1 - fill data for plane 1
  D0 - fill data for plane 0

Index 1 - Set/Reset Enable Register
  D7-D4 - reserved (0)
  D3 - enable Set/Reset for plane 3 (1 = enable)
  D2 - enable Set/Reset for plane 2
  D1 - enable Set/Reset for plane 1
  D0 - enable Set/Reset for plane 0
Index 2 - Color Compare Register
  D7-D4 - reserved
  D3 - Color Compare value for plane 3
  D2 - Color Compare value for plane 2
  D1 - Color Compare value for plane 1
  D0 - Color Compare value for plane 0
Index 3 - Data Rotate/Function Select Register
  D7-D5 - reserved (0)
  D4,D3 - Function Select
  D2-D0 - Rotate Count
  D4 D3 - Logical Operation
    0 0 Write data unmodified
    0 1 Write data AND processor latches
    1 0 Write data OR processor latches
    1 1 Write data XOR processor latches
Index 4 - Read Plane Select Register
  D7-D2 - reserved (0)
  D1,D0 - defines color plane for reading (0-3)
Index 5 - Mode Register
  D7 - reserved (0)
  D6 - 256 color mode
  D5 - Shift Register Mode
  D4 - Odd/Even Mode
  D3 - Color Compare Mode Enable (1 = enable)
  D2 - reserved (0)
  D1,D0 - Write Mode
    0 0 Direct write (Data Rotate, Set/Reset may apply)
    0 1 Use processor latches as write data
    1 0 Color plane n (0-3) is filled with the
      value of bit n in the write data
    1 1 Use (rotated) write data ANDed with Bit Mask as Bit Mask
      Use Set/Reset as if Set/Reset was enabled for all planes

Index 6 - Miscellaneous Register
D4 to D7 - Reserved
D2 to D3 - Memory Map
D3 D2
0 0 A000 for 128k
0 1 A000 for 64k
1 0 B000 for 32k
1 1 B800 for 32k
D1 - Chain odd planes to even
D0 - Graphics mode (disable character generator)
Index 7 - Color Don't Care Register
D7-D4 - reserved (0)
D3 - Plane 3 don't care
D2 - Plane 2 don't care
D1 - Plane 1 don't care
D0 - Plane 0 don't care
Index 8 - Bit Mask Register
D7 - mask data bit 7
D6 - mask data bit 6
D5 - mask data bit 5
D4 - mask data bit 4
D3 - mask data bit 3
D2 - mask data bit 2
D1 - mask data bit 1
D0 - mask data bit 0

## Attribute Controller - 3C0h, 3C1h

Index 00 to 0Fh - The Palette Registers
D6,D7 - Reserved
D0 to D5 - Color value
Index 10h - Mode Control Register
D7 - P4,P5 Source Select
D6 - Pixel Width
D5 - Horizontal Panning Compatibility
D4 - reserved
D3 - Background Intensify/Enable BLink
D2 - Line graphics enable (text modes only)
D1 - Display Type
D0 - Graphics/Text Mode
Index 11h - Screen Border Color

Index 12h - Color Plane Enable Register
D7,D6 - reserved
D5,D4 - Video Status Mux
D3 - Enable Color Plane 3
D2 - Enable Color Plane 2
D1 - Enable Color Plane 1
D0 - Enable Color Plane 0
Index 13 - Horizontal Panning Register
D7-D4 - reserved
D3-D0 - Horizontal Pan

| **Value** | **Number of pixels shifted to the left 0+, 1+, 2+ + 3+, 7, 7+** | **13h** | **Other modes** |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 1 | 2 | 1 | |
| 2 | 3 | 2 | 1 |
| 3 | 4 | 3 | |
| 4 | 5 | 4 | 2 |
| 5 | 6 | 5 | |
| 6 | 7 | 6 | 3 |
| 7 | 8 | 7 | |
| 8 | 9 | | |

Index 14 - Color Select Register
D7-D4 - reserved
D3 - color 7
D2 - color 6
D1 - color 5
D0 - color 4

## 3C6 - Pixel Mask Register

## 3C7 - DAC State Register (Read Only)

## 3C7 - Look-up Table Read Index (Write Only)

## 3C8 - Look-up Table Write Index

## 3C9 - Look-up Table Data Register

# C

# Character Set

# Character Set

| | DEC | HEX | | DEC | HEX | | DEC | HEX | | DEC | HEX | | DEC | HEX | | DEC | HEX | | DEC | HEX | | DEC | HEX |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | | 32 | 20 | @ | 64 | 40 | ` | 96 | 60 | Ç | 128 | 80 | á | 160 | A0 | └ | 192 | C0 | α | 224 | E0 |
| ☺ | 1 | 1 | ! | 33 | 21 | A | 65 | 41 | a | 97 | 61 | ü | 129 | 81 | í | 161 | A1 | ┴ | 193 | C1 | ß | 225 | E1 |
| ☻ | 2 | 2 | " | 34 | 22 | B | 66 | 42 | b | 98 | 62 | é | 130 | 82 | ó | 162 | A2 | ┬ | 194 | C2 | Γ | 226 | E2 |
| ♥ | 3 | 3 | # | 35 | 23 | C | 67 | 43 | c | 99 | 63 | â | 131 | 83 | ú | 163 | A3 | ├ | 195 | C3 | π | 227 | E3 |
| ♦ | 4 | 4 | $ | 36 | 24 | D | 68 | 44 | d | 100 | 64 | ä | 132 | 84 | ñ | 164 | A4 | ─ | 196 | C4 | Σ | 228 | E4 |
| ♣ | 5 | 5 | % | 37 | 25 | E | 69 | 45 | e | 101 | 65 | à | 133 | 85 | Ñ | 165 | A5 | ┼ | 197 | C5 | σ | 229 | E5 |
| ♠ | 6 | 6 | & | 38 | 26 | F | 70 | 46 | f | 102 | 66 | å | 134 | 86 | ª | 166 | A6 | ╞ | 198 | C6 | µ | 230 | E6 |
| • | 7 | 7 | ' | 39 | 27 | G | 71 | 47 | g | 103 | 67 | ç | 135 | 87 | º | 167 | A7 | ╟ | 199 | C7 | τ | 231 | E7 |
| ◘ | 8 | 8 | ( | 40 | 28 | H | 72 | 48 | h | 104 | 68 | ê | 136 | 88 | ¿ | 168 | A8 | ╚ | 200 | C8 | Φ | 232 | E8 |
| ○ | 9 | 9 | ) | 41 | 29 | I | 73 | 49 | i | 105 | 69 | ë | 137 | 89 | ⌐ | 169 | A9 | ╔ | 201 | C9 | Θ | 233 | E9 |
| ◙ | 10 | A | * | 42 | 2A | J | 74 | 4A | j | 106 | 6A | è | 138 | 8A | ¬ | 170 | AA | ╩ | 202 | CA | Ω | 234 | EA |
| ♂ | 11 | B | + | 43 | 2B | K | 75 | 4B | k | 107 | 6B | ï | 139 | 8B | ½ | 171 | AB | ╦ | 203 | CB | δ | 235 | EB |
| ♀ | 12 | C | , | 44 | 2C | L | 76 | 4C | l | 108 | 6C | î | 140 | 8C | ¼ | 172 | AC | ╠ | 204 | CC | ∞ | 236 | EC |
| ♪ | 13 | D | - | 45 | 2D | M | 77 | 4D | m | 109 | 6D | ì | 141 | 8D | ¡ | 173 | AD | ═ | 205 | CD | φ | 237 | ED |
| ♫ | 14 | E | . | 46 | 2E | N | 78 | 4E | n | 110 | 6E | Ä | 142 | 8E | « | 174 | AE | ╬ | 206 | CE | ε | 238 | EE |
| ☼ | 15 | F | / | 47 | 2F | O | 79 | 4F | o | 111 | 6F | Å | 143 | 8F | » | 175 | AF | ╧ | 207 | CF | ∩ | 239 | EF |
| ► | 16 | 10 | 0 | 48 | 30 | P | 80 | 50 | p | 112 | 70 | É | 144 | 90 | ░ | 176 | B0 | ╨ | 208 | D0 | ≡ | 240 | F0 |
| ◄ | 17 | 11 | 1 | 49 | 31 | Q | 81 | 51 | q | 113 | 71 | æ | 145 | 91 | ▒ | 177 | B1 | ╤ | 209 | D1 | ± | 241 | F1 |
| ↕ | 18 | 12 | 2 | 50 | 32 | R | 82 | 52 | r | 114 | 72 | Æ | 146 | 92 | ▓ | 178 | B2 | ╥ | 210 | D2 | ≥ | 242 | F2 |
| ‼ | 19 | 13 | 3 | 51 | 33 | S | 83 | 53 | s | 115 | 73 | ô | 147 | 93 | │ | 179 | B3 | ╙ | 211 | D3 | ≤ | 243 | F3 |
| ¶ | 20 | 14 | 4 | 52 | 34 | T | 84 | 54 | t | 116 | 74 | ö | 148 | 94 | ┤ | 180 | B4 | ╘ | 212 | D4 | ⌠ | 244 | F4 |
| § | 21 | 15 | 5 | 53 | 35 | U | 85 | 55 | u | 117 | 75 | ò | 149 | 95 | ╡ | 181 | B5 | ╒ | 213 | D5 | ⌡ | 245 | F5 |
| ▬ | 22 | 16 | 6 | 54 | 36 | V | 86 | 56 | v | 118 | 76 | û | 150 | 96 | ╢ | 182 | B6 | ╓ | 214 | D6 | ÷ | 246 | F6 |
| ↨ | 23 | 17 | 7 | 55 | 37 | W | 87 | 57 | w | 119 | 77 | ù | 151 | 97 | ╖ | 183 | B7 | ╫ | 215 | D7 | ≈ | 247 | F7 |
| ↑ | 24 | 18 | 8 | 56 | 38 | X | 88 | 58 | x | 120 | 78 | ÿ | 152 | 98 | ╕ | 184 | B8 | ╪ | 216 | D8 | ° | 248 | F8 |
| ↓ | 25 | 19 | 9 | 57 | 39 | Y | 89 | 59 | y | 121 | 79 | Ö | 153 | 99 | ╣ | 185 | B9 | ┘ | 217 | D9 | ∙ | 249 | F9 |
| → | 26 | 1A | : | 58 | 3A | Z | 90 | 5A | z | 122 | 7A | Ü | 154 | 9A | ║ | 186 | BA | ┌ | 218 | DA | · | 250 | FA |
| ← | 27 | 1B | ; | 59 | 3B | [ | 91 | 5B | { | 123 | 7B | ¢ | 155 | 9B | ╗ | 187 | BB | █ | 219 | DB | √ | 251 | FB |
| ∟ | 28 | 1C | < | 60 | 3C | \ | 92 | 5C | \| | 124 | 7C | £ | 156 | 9C | ╝ | 188 | BC | ▄ | 220 | DC | ⁿ | 252 | FC |
| ↔ | 29 | 1D | = | 61 | 3D | ] | 93 | 5D | } | 125 | 7D | ¥ | 157 | 9D | ╜ | 189 | BD | ▌ | 221 | DD | ² | 253 | FD |
| ▲ | 30 | 1E | > | 62 | 3E | ^ | 94 | 5E | ~ | 126 | 7E | ₧ | 158 | 9E | ╛ | 190 | BE | ▐ | 222 | DE | ■ | 254 | FE |
| ▼ | 31 | 1F | ? | 63 | 3F | _ | 95 | 5F | ⌂ | 127 | 7F | ƒ | 159 | 9F | ┐ | 191 | BF | ▀ | 223 | DF | | 255 | FF |

# D

# *Standard VGA Modes*

# Standard VGA Modes

| Mode | Type | Resolution | Colors | Display Type |
|---|---|---|---|---|
| 0,1 | Text | 40 cols x 25 rows, 8x8 cell | 16 | CGA |
| 0* | Text | 40 cols x 25 rows, 8x14 cell | 16 | EGA |
| 0+ | Text | 40 cols x 25 rows, 9x16 cell | 16 | VGA |
| 2,3 | Text | 80 cols x 25 rows, 8x8 cell | 16 | CGA |
| 2* | Text | 80 cols x 25 rows, 8x14 cell | 16 | EGA |
| 2+,3+ | Text | 80 cols x 25 rows, 9x16 cell | 16 | VGA |
| 4,5 | Graphics | 320 horizontal x 200 vertical | 4 | CGA |
| 6 | Graphics | 640 horizontal x 200 vertical | 2 | CGA |
| 7 | Text | 80 cols x 25 rows, 8x14 cell | Mono | MDA |
| 7+ | Text | 80 cols x 25 rows, 9x16 cell | Mono | VGA |
| D | Graphics | 320 horizontal x 200 vertical | 16 | CGA |
| E | Graphics | 640 horizontal x 200 vertical | 16 | CGA |
| F | Graphics | 640 horizontal x 350 vertical | Mono | MDA |
| 10h | Graphics | 640 horizontal x 350 vertical | 16 | EGA |
| 11h | Graphics | 640 horizontal x 480 vertical | 2 | VGA |
| 12h | Graphics | 640 horizontal x 480 vertical | 16 | VGA |
| 13h | Graphics | 320 horizontal x 200 vertical | 256 | VGA |

Display types:

| | |
|---|---|
| CGA | - Color display (TTL, 640x200, 15.75kHz) |
| EGA | - Enhanced display (TTL, 640x350, 21.85kHz) |
| VGA | - VGA display (Analog, 720x400 & 640x480, 31.5kHz) |

# E

# *Examples Summary*

# Examples Summary

# F

# *VGA Boards*

# SuperVGA Mode Summary

| Manufacturer and Model | | Chip | Text 132x44 132x43 | Graph 640x400 256-col | Graph 640x480 256-col | Graph 800x600 256-col | Graph 1024x768 256-col | Graph 800x600 16-col | Graph 1024x768 16-col | Graph 1024x768 4-col |
|---|---|---|---|---|---|---|---|---|---|---|
| Ahead | Wizard | Ahead | 22h | 60h | 61h | 62h | 63h i | 6Ah | 74h | 75h |
| ATI | VGAWONDER | ATI | 33h | 61h | 62h | 63h | | 6Ah | 55h/65h | 67h |
| AST | VGA Plus | WD | 56h | 5Eh | 5Fh | | | 58h | | |
| Boca Res. | 1024 VGA | Chips | | 78h | 79h | | | 6Ah | 72h | |
| Chips & Technologies | | Chips | | 78h | 79h | | | 6Ah | 72h | |
| Cirrus | | Cirrus | 20h | 50h | | | | 6Ah | | |
| Everex | Viewpoints | Trident | *0Bh | *14h | *30h | *31h | | *02h | *20h | *60h |
| Genoa | 6400 | Genoa | 63h | 7Eh | 5Ch | 5Eh | | 79h | 5Fh | 7Fh |
| Headland | VRAM | Headland | *42h | *66h | *67h | *69h | | *62h | *65h | *64h |
| Headland | VGA 1024i | Headland | *42h | *66h | *67h | | | 6Ah | *65h | *64h |
| HP | | Paradise | | 5Eh | 5Fh | | | 58h | | |
| MaxLogic | MaxVGA | Cirrus | 20h | 50h | | | | 6Ah | | |
| NSI | VGA 16 | NSI | 22h | | 2Eh | 30h | | 6Ah | 37h | 38h (2 col) |
| Orchid | ProDesigner | Tseng | 24h | | 2Eh | 30h | | 29h | 37h | |
| Quadram | Spectra | Tseng | 22h | | 2Eh | 30h | | 29h | 37h i | |
| Sigma | VGA Legend | ET-4000 | 1Dh | | 2Eh | 30h | 38h | 6Ah | 37h i | |
| Sota | VGA 16 | Tseng | 22h | 1Bh | 2Eh | 30h | | 29h | 37h | |
| STB | VGA Extra | Tseng | 22h | | 2Eh | 30h | | 29h | 37h | |
| Tecmar | VGA ADGM | Tseng | #17h | 1Bh | 1Ch | 1Dh | | 16h | 19h | |
| TrueTech | Hires VGA | Zymos | 57h | 5Ch | 5Dh | 5Eh | | 6Ah | 5Fh | |
| Tseng | | Tseng | 22h | | 2Eh | 30h | | 29h | 37h | |
| VESA | | | | *100h | *101h | *103h | *105h | *102h | *104h | |
| WD | Professional | WD | 54h | 5Eh | 5Fh | | | 58h | | |
| WD | 1024 | WD | 54h | 5Eh | 5Fh | 5Ch | | 58h | 5Dh | 5Bh |
| Willow Publishers VGA | | Tseng | 22h | | 2Eh | 30h | | 29h | 37h i | |
| Zymos | | Zymos | 57h | 5Ch | 5Dh | 5Eh | | 6Ah | 5Fh | |

i = Interlaced only
* = Needs special mode select call for INT 10h
# = Must use BIOS functions AH = 12h, BL = 30 to set scan line count, and AH = 11 to select font
ET-4000 = New chip from Tseng. Paging as follows. Read = 3CDh bits 0-3. Write = 3CDh bits 4-7

# Addresses of companies covered in this book

**Ahead Systems Inc.**
44244 Fremont Boulevard
Fremont, CA 94538
Tel: (415) 623–0900

**ATI Technologies Inc.**
3761 Victoria Park Avenue
Scarborough, Ontario
Canada M1W 3S2
Tel: (416) 756–0718
Fax: (416) 756–0720

**Boca Research Inc.**
6401 Congress Avenue
Boca Raton, FL 33487
Tel: (407) 997–6227
Fax: (407) 997–0918

**Chips and Technologies Inc.**
3050 Zanker Road
San Jose, CA 95134
(408) 434–0600

**Cirrus Logic Inc.**
1463 Center Pointe Drive
Milpitas, CA 95035
Tel: (408) 945–8300
Fax: (408) 263–5682

**Everex Systems Inc.**
48431 Milmont Drive
Fremont, CA 94538
Sales: (415)683–2100
Tech: (415) 498–1115
FAX: (415) 651–0728
BBS: (415) 683–2924

**Genoa Systems Corporation**
75 E. Trimble Road
San Jose, CA 95131
Tel: (408) 432–9090
Fax: (408) 434–0997

**Headland Technology Inc.**
(formerly Video Seven)
46221 Landing Parkway
Fremont, CA 94538
Tel: (415) 656–7800
(800) 248–1850
(800) 553–1850 (Calif.)
Fax: (415) 657–4604
BBS: (415) 656–0503

**MaxLogic Systems Inc.**
48350 Milmont Drive
Fremont, CA 94538
Tel: (415) 683–2684
Tech: (415) 490–4199

**Paradise**
see Western Digital

**STB Systems Inc**
1651 N. Glenville
P.O. Box 850957
Richardson, TX 75085
Tel: (214) 234–8750

**Trident Microsystems Inc.**
321 Soquel Way
Sunnyvale, CA 94086
Tel: (408) 738–3194
Fax: (408) 738–0905

**TrueTech Inc.**
181-B W. Orangethorpe
Placentia, CA 92670
Tel: (714) 961–0438
(800) PC–AT–386
Fax: (714) 961–0952

**Tseng Laboratories Inc.**
10 Pheasant Run
Newtown Commons
Newtown, PA 18940
Tel: (215) 968–0502
Fax: (215) 860–7713

**VESA**
1330 South Bascom Avenue, Suite D
San Jose, CA 95128
Tel: (408) 971–7525
Fax: (408) 286–8988

**Video Seven**
see Headland

**Western Digital Imaging**
800 E. Middlefield Road
Mountain View, CA 94043
Tel: (415) 960–3360
(800) 356–5787 (outside Calif.)
BBS: (415)–968–1834

**ZyMOS Corporation**
477 N. Mathilda Avenue
Sunnyvale, CA 94086
Tel: (408) 730–5400
Fax: (408) 730–5473

# VGA manufacturers

**Ahead Systems Inc.**
44244 Fremont Boulevard
Fremont, CA 94538

**American Mitac**
410 East Plumeria Drive
San Jose, CA 95134

**AST Research**
2121 Alton
Irvine, CA 92714

**ATI Technologies Inc.**
3761 Victoria Park Avenue, Scarborough
Ontario, Canada M1W 3S2

**Boca Research Inc.**
6401 Congress Avenue
Boca Raton, FL 33487

**Cardinal Technologies**
1827 Freedom Road
Lancaster, PA 17601

**C2 Micro Systems**
1205 Fulton Place
Fremont, CA 94539

**Chips and Technologies Inc.**
3050 Zanker Road
San Jose, CA 95134

**Cirrus Logic Inc.**
1463 Center Pointe Drive
Milpitas, CA 95035

**Club American Technologies**
3401 W. Warren Avenue
Fremont, CA 94539

**Colorgraphic Communications Corp**
5388 Peachtree Road, P.O. Box 80448
Atlanta, GA 30366

**Commax Technologies**
2031 Concourse Drive
San Jose, CA 95131

**Compaq Computer**
20555 FM 149, P.O. Box 692000
Houston, TX 77269

**Computer Electronik Infosys of America Inc.**
512A Herndon Parkway
Herndon, VA 22070

**CSS Labs**
1641 McGaw Avenue
Irvine, CA 92714

**Dell Computer Corp.**
9505 Arboretum Boulevard
Austin, TX 78759

**Enertronics**
1910 Pine Street
St. Louis, MO 63103

**Everex Systems Inc.**
48431 Milmont Drive
Fremont, CA 94538

**Genoa Systems Corporation**
75 E. Trimble Road
San Jose, CA 95131

**Goldstar Technology**
1130 E. Arques Avenue
Sunnyvale, Ca. 94086

**GVC-Chenel Corp.**
99 Demarest Road
Sparta, N.J. 07871

**Headland Technology Inc.**
(formerly Video Seven)
46221 Landing Parkway
Fremont, CA 94538

**Hewlett-Packard**
19310 Pruneridge Avenue
Cupertino, CA 95014

**Hercules Computer Technology**
921 Parker Street
Berkeley, CA 94710

**IBM**
900 King Street
1A515 Rye Brook, NY 10573

**Imagraph**
800 W. Cummings Park
Woburn, Mass 01801

**Intel**
would not give address
call (800) 548–4725

**Intelligent Graphics**
4800 Great American Parkway, Suite 200
Santa Clara, CA 95054

**Logitech**
6505 Kaiser Drive
Fremont, CA

**Magni Systems**
9500 SW Gemini Drive
Veaverton, OR 97005

**MaxLogic Systems Inc.**
48350 Milmont Drive
Fremont, CA 94538

**Micron Technology**
2805 E. Columbia Road
Boise, Idaho, 83706

**MicroWay Inc**
P.O. Box 79
Kingston, Mass 02364

**Mylex**
47650 Westinghouse Drive
Fremont, CA 94539

**National Semiconductor**
(formerly Quadram)
750 Central Expressway, m/2 3410
Santa Clara, CA 95050

**NSI Logic**
Cedar Hill Business Park
257-B Cedar Hill Road
Marlboro, Mass 01752

**Orchid Technology**
45365 Northport Loop West
Fremont, CA 94538

**Paradise**
see Western Digital

**Personal Computer Graphics Corp**
5819 Uplander Way
Culver City, CA 90230

**Prism Imaging Systems**
5309 Randall Place
Fremont, CA 94538

**Quadram**
see National Semiconductor

**QDP Computer Systems Inc**
23632 Mercentile Road
Beachwood, Ohio 44122

**Relisys**
320 S. Milpitas Boulevard
Milpitas, Ca. 95035

**Renaissance GRX**
Cedar Park
2265 116th Avenue, N.E.m
Bellvue, WA 98004

**Scouri/Twinhead Corp.**
P.O. Box 702, 30 Chapin Road
Pine Brook, NJ 07058

**SCOA Systems**
2100 Golf Road, Suite 100
Rolling Medows, IL 60008

**Sigma Design**
46501 Landing Parkway
Fremong, CA 94538

**SMT**
1145 Linda Vista Drive
San Marcos, CA 92069

**Sota Technology**
559 Weddell Drive
Sunnyvale, CA 94089

**STB Systems Inc**
1651 N. Glenville, P.O. Box 850957
Richardson, TX 75085

**Tatung Co. of America**
2850 El Presidio Street
Long Beach, CA 90810

**Tecmar**
6225 Cochran Road
Solon, OH 44139

**Thomson Consumer Products Corp**
5731 West Slauson Avenue, Suite 111
Culver City, CA 90230

**Trident Microsystems Inc.**
321 Soquel Way
Sunnyvle, CA 94086

**TrueTech Inc.**
181-B W. Orangethorpe
Placentia, CA 92670

**Tseng Laboratories Inc.**
10 Pheasant Run, Newtown Commons
Newtown, PA 18940

**US Video**
One Stamford Landing, 62 Southfield Avenue
Stamford, CT 06902

**Video Seven**
see Headland

**Vutek**
10855 Sorrento Valley Road
San Diego, CA 92121

**Western Digital Imaging**
(fromerly Paradise)
800 E. Middlefield Road
Mountain View, CA 94043

**Willow Peripherals**
190 Willow Ave
Bronx, NY 10454

**Zenith Data Systems**
100 Milwaukee Avenue,
Glenview, IL 60025

**ZyMOS Corporation**
477 N. Mathilda Avenue
Sunnyvale, CA 94086

# G

# *VGA Displays*

# VGA Display Manufacturers

**Acer Technologies Corp.**
401 Charcot Avenue
San Jose, Ca. 95131
(408)922-0333
(800)538-1542
FAX:(408)922-0176

**Amdek Corp.**
3471 N. First Street
San Jose, Ca. 95134
(408)559-3953 Sandy Parker
(800) PC-AMDEK
FAX:(408)922-5729

**Aydin Controls**
414 Commerce Drive
Fort Washington, Pa. 19034
(215)542-7800
(800)366-8889

**Conrac Corp.**
1724 S. Mountain Avenue
Duarte, Ca. 91010
(818)303-0095
FAX:(818)303-5484
Ruby

**CTX International Inc.**
161 Commerce Way
Walnut, Ca. 91789
(714)595-6146
FAX:(714)595-6293

**Electrohome Ltd.**
809 Wellington Street North
Kitchener, Ontario
N2G 4J6 Canada
(519)744-7111

**Goldstar Technology, Inc.**
1130 E. Arques Avenue
Sunnyvale, Ca. 94086
(408)432-1331 Bill Lynch
FAX:(408)739-0202

**Hitachi America, Ltd.**
950 Elm Avenue
San Bruno, Ca. 94066
(415)589-8300

**Idek America Inc.**
204 S. Olive Street
Rolla, Mo. 65401
(314)364-7500

**Magnavox**
NAP Consumer Electronics Group
P.O. Box 14810
Knoxville, Tenn. 37914
(615)521-4316

**Mitsubishi Electronics of America, Inc.**
991 Knox Street
Torrance, Ca. 90502
(800)441-2345 ext. 54M
(213)217-5732
(213)515-3993
FAX:(213)324-6466

**Nanao USA Corp.**
23510 Telo Avenue, Suite 5
Torrance, Ca. 90505
Steven Leon—Answering Machine
(213)670-5606
(213) 325-5202
FAX: (213) 530-1679

**NEC Home Electronics**
Computer Products Division
1255 Michael Drive
Wood Dale, Illinois 60191-1094
(312)860–9500
(800) 826–2255
(800) FONE--NEC
(800) 323–1728

**Packard Bell**
9425 Canoga Avenue
Chatsworth, Ca. 91311
(818)733–4400
(800)521–7979
FAX:(818)773–9521

**Panasonic**
2 Panasonic Way, 7D-3
Secaucus, N.J. 07094
(201)348–7000
(800)PIC–8086

**Princeton Graphic Systems**
601 Ewing Street Building A
Princeton, N.J. 08540
(609)683–1660

**Princeton Graphics Systems**
1100 Northmeadow Parkway
Roswell, Ga. 30076
(404)475–2725
(800)241–3946
FAX:(404)475–2707

**Quimax Systems, Inc.**
844 Del Rey Avenue
Sunnyvale, Ca. 94086
(408)773–8282
FAX: (408)730–2340

**Relisys**
320 S. Milpitas Boulevard
Milpitas, Ca. 95035
(408)945–9000
(408)945–1062
FAX:(408)945–0587

**Sampo Corp. of America**
6350 Peachtree Industrial Boulevard
Norcross, Georgia 30071
(404) 449–6220
FAX: (404) 447–1109

**Samsung Electronics America Inc.**
301 Mayhill Street
Saddlebrook, NJ 07662
(201)587–9600
FAX:(201)587–9178

**Seiko Instruments USA**
1144 Ringwood Court
San Jose, Ca. 95131
(800)888–0817
(408)943–9100
(800)553–5315
(408)922–5900
FAX:(408)922–5835

**Sony Corp. of America**
1 Sony Drive
Park Ridge, N.J. 07656
(201)930–1000
(800)222–SONY

**Tatung of America Inc.**
2850 El Presidio Street
Long Beach, Ca. 90810
(213)637–2105
(213)979–7055
(800)421–2929
FAX:(213)637–8484

**Taxan USA Corp.**
161 Nortech Parkway
San Jose, Ca. 95134
(800)544-3888
(408)946-3400
FAX:(408)262-9059

**Taxan USA Corp.**
18005 Cortney Court
City of Industry, Ca. 91748
(818)810-1291

**Thompson Consumer Products Corp.**
5731 West Slauson Avenue, Suite 111
Culver City, Ca. 90230
(800)237-9483

**Zenith Data Systems**
1000 Milwaukee Avenue
Glenview, Ill. 60025
(800)842-9000

# H

# *Debugging Video*

# Debugging Video

Some special precautions are necessary when debugging programs that interface with a video display adapter.

Most software debugging tools output debugging information to the system console device. If the software being debugged is also outputting information to the console, the resulting interaction between the program and the debugger may produce totally unpredictable results. The debugger itself may even cease to function.

One way around this problem is to install a secondary display adapter in the system. If the program under debug outputs to the secondary display, it will not interfere with the debugger's use of the primary (console) display. If, for instance, the software under debug is outputting color graphics to an EGA, an MDA monochrome adapter can be installed. The MODE MONO command can be used to declare the monochrome display as the console device for the debugger. This method is not guaranteed to eliminate problems if the program under debug uses any DOS or BIOS functions that output to the console device. For SYMDEB and CodeView debuggers you can use SYMDEB /M or CV /2 to redirect I/O to secondary (monochrome) monitor.

Another approach is to connect a CRT terminal to a serial port, then redirect console I/O to the serial port by using the CTTY COM1 command. This method also may not be reliable if the software being debugged outputs data to the console device. For SYMDEB debugger you can use =COM1 (after the debugger was invoked) to redirect I/O to serial port 1.

In cases where only one display adapter is available, or if the debugger and program under debug must both output to the console device, special precautions must be taken. Program single stepping or setting of breakpoints must be done with great care. When the debugger is activated by a breakpoint, it will output data to the console, thereby altering the state of the display adapter. This may destroy any adapter configuration that was done by the program under debug.

For EGA and VGA, register access is usually a two step process of outputting first an index and then register data. If a breakpoint is set between the output of the index and data, it is virtually guaranteed that the debugger will overwrite the index and the subsequent data output will be performed incorrectly.

Above all, bear in mind that if the display adapter is left in an invalid state for more than a few seconds damage to the display may result.

# Glossary

# Glossary

**40x25** Text mode of operation which displays 25 lines of text with 40 character columns per line.

**80x25** Text mode of operation which displays 25 lines of text with 80 character columns per line.

**80X86** The Intel family of microprocessors, including the 8086, 8088, 80186, 80286, and 80386, which are all software compatible.

**320x200, 640x350, etc.** Graphics screen resolutions, expressed as the number of pixels displayed horizontally across the screen by the number of pixels displayed vertically; i.e., 320x200 means 320 horizontal pixels by 200 vertical pixels.

**ADAPTER or DISPLAY ADAPTER** A circuit board designed to interface a display device to a computer system, such as MDA, CGA, EGA, or VGA.

**ALL POINTS ADDRESSABLE (APA)** IBM terminology for a graphics mode, so called because each dot on the display screen may be controlled independently.

**ANALOG INTERFACE** A type of interface used between video controller and video display in which colors are determined by the voltage levels on three output lines, normally called the RED, GREEN, and BLUE (or RGB) lines. A theoretically unlimited number of colors can be supported by this method. Output voltage normally varies between zero volts (for black) to one volt (for maximum brightness). Load impedance is normally 75 ohms.

**ANALOG DISPLAY/ANALOG MONITOR** A display device that uses an analog interface. Such displays use variable voltage and allow a large number of colors.

**ASCII** American Standard Code for Information Interchange, the most common method of digitally encoding alphanumeric data.

**ASPECT RATIO** The ratio of height to width of a single pixel on a display screen. High resolution displays usually have a 1:1 aspect ratio, or are said to have SQUARE PIXELS. Graphics drawing algorithms must compensate for the aspect ratio of the display if it is not 1:1; otherwise, circles will appear elliptical and squares will appear rectangular.

**ATTRIBUTE CONTROLLER** The section of logic on EGA and VGA which generates display attributes (see DISPLAY ATTRIBUTES).

**AUTOSENSE** Capability of display adapter to automatically detect 16 bit connection and operate BIOS in 16 bit mode.

**AUTOSIZING** Capability of a display to automaticaly adjust size of the displayed when switching from one resolution to another. Some displays will autosize only for a fixed set of frequencies, and need manual adjustment for the rest.

**BACKGROUND** In text mode, the area of a character cell that is not illuminated by the character. The rest of the character cell is referred to as the FOREGROUND. In graphics mode, the area of the screen that is not illuminated by a graphics object.

**BIOS/ROM BIOS** Basic Input Output System; in IBM compatible personal computers, this is a set of ROM based firmware routines which control the resources of the system and make them available to applications programs in an orderly fashion.

**BIOS DATA AREA** An area in system memory where the EGA/VGA BIOS stores data defining the display resolution, cursor position, etc.

**BITBLT** Bit oriented BLock Transfer; this is a type of graphics drawing routine which moves a rectangle of display data from one area of display memory to another. This can be difficult because the data to be moved is usually neither contiguous nor byte aligned. Graphics controllers frequently include varying levels of hardware assist to help speed BITBLT operations.

**BIT MAPPED GRAPHICS** A graphics display mode in which each pixel on the display surface is represented by one or more bits in display memory. All EGA and VGA graphics modes are bit mapped.

**BIT PLANE** See Color Plane.

**BLANKING, BLANK PULSE** For CRT displays, a timing signal which shuts off the electron beam during retrace intervals to prevent unwanted diagonal lines of light from appearing on the screen.

**BLOCK GRAPHICS (OR LINE GRAPHICS)** In text modes, a set of primitive graphics objects that can be used as text characters to create simple graphics such as borders and lines.

**CGA** Color Graphics Adapter, the first IBM color graphics product for personal computers (EGA was the second.) CGA can produce 4 color graphics or 16 color text at a resolution of 640 pixels horizontally by 200 pixels vertically.

**CHARACTER CELL** In text mode, the area of display used to display one character. On EGA, character cells are either 8 or 9 pixels wide and usually either 8, 14 or 16 pixels high.

**CHARACTER CODE** A one byte code representing a text character (usually ASCII).

**CHARACTER GENERATOR** A translation table used to translate an ASCII character code into character font information for display. Some display adapters use ROM based character generators; for EGA and VGA, the character generator is loaded into a section of display RAM.

**CHARACTER SET** The set of characters which a display adapter is capable of displaying. In text mode, this is determined by the contents of the Character Generator. The EGA character set contains 256 characters.

**COLOR PALETTE** The set of colors that are available with a given display system.

**COLOR PLANES** In plane oriented graphics adapters, color planes are overlapping pages or sections of memory which control different display colors.

**COMPOSITE DISPLAY** A display device that uses a composite sync signal (combined horizontal and vertical sync) as opposed to separate sync signals.

**CONSOLE DEVICE** The keyboard and display that are used to control the computer. In multiple display systems, the console device can usually be assigned to be any one of the display devices.

**CPU** Central Processing Unit, another name for the system processor.

**CRT DISPLAY** Cathode Ray Tube Display; all of the the display devices discussed in this book fall into this category.

**CRT CONTROLLER (CRTC)** On the EGA and VGA, as well as many other video display adapters, the CRT Controller is the circuit which is responsible for generating the timing signals required to operate a CRT display (including blanking and retrace sync pulses.)

**DIGITAL INTERFACE** A type of interface used between video controller and video display in which display color is controlled by digital color control lines switching on and off. The number of colors that can be supported depends on the number of signal lines in the interface. Most digital interfaces are TTL (Transistor-Transistor Logic) compatible. CGA and EGA both use digital interfaces.

**DIGITAL DISPLAY** A display device that uses a digital interface, where limited number (2 for VGA displays) of discrete voltages is used to control small set of colors (4 for VGA displays) for each color input.

**DISPLAY ATTRIBUTE** A programmable display characteristic. In graphics modes, color is usually the only display attribute. In text modes, attributes may include blinking, underlining, or reverse video.

**DISPLAY REFRESH (or SCREEN REFRESH)** An image drawn on a CRT display will only remain visible for a few milliseconds (the presistence of the screen phosphor), unless it is redrawn continuously. This process is called DISPLAY REFRESH or SCREEN REFRESH. Different displays use different refresh rates, but display refresh is normally required between 50 and 70 times a second to avoid any visible screen flickering. 60 times a second is a common refresh rate.

**DOT CLOCK (or PIXEL CLOCK)** The timing signal on a display adapter that controls the serial output of pixels to the display device.

**DOUBLE SCAN** A technique used by VGA to gain compatibility with the lower resolution CGA. Each horizontal scan line is drawn twice, which converts a CGA 200 line image into a

VGA 400 line image. This also partially compensates (actually over-compensates) for the different aspect ratio of the VGA.

**DRIVER** A software module that interfaces a particular display device to an application program. EGA drivers have been written for programs such as Microsoft Windows, DRI GEM, Lotus 1-2-3, etc.

**ELECTRON BEAM** In a CRT display, a moving beam of electrons creates the display image seen on the display screen. Timing and modulation of the electron beam are controlled by the display adapter.

**EMULATION** A technique for making one type of display device appear to operate as if it were a different display device. Emulations improve the usefulness of a product by making it compatible with other products. EGA is capable of emulating MDA and sometimes CGA and Hercules. VGA is capable of emulating EGA, CGA and MDA.

**FEATURE CONNECTOR** An expansion connector on the EGA which can be used to combine other video signals with EGA video output. It is not widely used.

**FONT** This term originated in the publishing industry. A font is a character set of one particular size and style (such as 14 point Helvetica).

**FOREGROUND** In text mode, the portion of a character cell that is illuminated by the character font (as opposed to BACKGROUND.)

**GRAPHICS CONTROLLER** On EGA and VGA, a section of circuitry that can provide hardware assist for graphics drawing algorithms by performing logical functions on data written to display memory.

**GRAPHICS MODE** A display mode in which all pixels on the display screen can be controlled independently to draw graphics objects (as opposed to TEXT MODE, in which only a pre-defined set of characters can be displayed.)

**HERCULES GRAPHICS** Graphics programs which are compatible with the monochrome graphics adapter produced by Hercules Corporation.

**HGC** Hercules monochrome Graphics Controller.

**HORIZONTAL RETRACE** In CRT displays, the time interval when the electron beam is being returned from the right side of the display screen to the left side of the display screen. The electron beam is turned off during this time (HORIZONTAL BLANKING.)

**IBM COLOR DISPLAY (CD)** The display device marketed by IBM for use with the CGA display adapter.

**IBM ENHANCED COLOR DISPLAY (ECD)** The display device marketed by IBM for use with the EGA display adapter.

**INDEX REGISTER** A register used to indirectly address other registers.

**INTERLACED DISPLAY/8514 DISPLAY** Type of display where beam scans odd lines first, and on next vertical scan, scans all even lines. Such displays are typically less expensived than non-interlaced displays.

**I/O REGISTER** A data register (either read only, write only, or read-write), which is mapped into the I/O space of the processor.

**LATCH** In electronics, a type of memory device that captures and holds several bits of data.

**LIGHT PEN** A device that allows an operator to input commands to the computer by placing the pen tip to a certain position on the display screen (such as touching an item on a menu.) The application software must be written to support the use of a light pen. Light pens have not become nearly as popular as mice or joysticks for this purpose.

**LINE GRAPHICS** See BLOCK GRAPHICS

**MDA** Monochrome Display Adapter; the original display adapter marketed by IBM for personal computers. MDA has no bit-mapped graphics capability.

**MONOCHROME DISPLAY** A one color display device; often referred to as a black and white display, even though the color used is often amber or green. Sometimes referred to as a two color display (the second color being black.)

**MONITOR** Another term for a CRT Display.

**MULTISYNC DISPLAY** A display marketed by NEC Corporation. The Multisync is EGA compatible, and also supports higher resolutions. Many displays operate only at a single horizontal scanning rate. The Multisync display can operate over a range of scanning frequencies and screen resolutions. This term, as is term MULTIFREQUNCY DISPLAY, is often used to describe displays capable of working at several frequencies including 640x350, 640x480 and 800x600 with 256 colors.

**MULTIFREQUENCY DISPLAY** Type of display that is capble of working at several differnt frequencies. Typically a class of PC displays that support resolutions of 640x350, 640x480 and 800x600.

**NON-INTERLACED DISPLAY** Type of display where all lines (even and odd) are displayed in one vertical scan. As opposed to interlaced display where all odd lines are displayed on one vertical scan, and all even lines are displayed on the next scan.

**PALETTE** The choice of available colors with a color graphics display system. The term PALETTE is sometimes used to refer to a color look-up table.

**PANNING** A technique by which the display screen is made to appear to be a viewport into a larger display, and then the viewport is moved around so that different areas of the display come into view.

**PEL** IBM terminology for a pixel.

**PGC or PGA** Professional Graphics Controller, a high resolution color display adapter sold by IBM. The PGC was not highly successful as a product.

**PIXEL** A single dot on the the display surface. The smallest independently programmable display element.

**PRIMARY DISPLAY** An IBM term for the console device; the display where DOS displays prompts and messages.

**RASTER** The left-to-right, top-to-bottom scanning pattern made on the screen by the electron gun in a CRT display.

**RESOLUTOIN** A measure of the quality of image that can be shown on a particular display; usually expressed as the number of pixels that can be displayed horizontally across the display screen by the number of pixels that can be displayed vertically on the display screen.

**RGB** A type of interface used with color displays which uses three color signals (Red, Green and Blue).

**SCAN LINE** One horizontal scan of the electron beam in a CRT display.

**SCROLLING** On a text display, the process of moving the displayed text up or down (usually up) on the display screen, normally to make room for new text to be displayed. This allows a large block of text to be viewed a small amount at a time. Scrolling is usually done in an upward direction one line at a time so that the text appears to roll smoothly upward on the screen.

**SECONDARY DISPLAY** An IBM term for a display device which is not the console device, but that may be used by an application program to display data.

**SEQUENCER** The section of circuitry on EGA and VGA that controls timing for the board. The sequencer also contains memory plane enabling and disabling functions.

**SERIALIZER** On display adapters, the section of circuitry that converts words of display refresh data into a serial bit stream to be output to the display.

**SET/RESET** A function on EGA and VGA (poorly named) that permits a fill pattern to be quickly written into display memory. The Set/Reset function is part of the Graphics Controller.

**SIMULTANEOUS COLORS** The number of colors in a display system that can be displayed on the screen at one time. This number is limited by the circuitry of the display adapter, and is usually much smaller than the number of colors the display device can actually support. The number of simultaneous colors a display adapter will support is normally determined by the number of color planes, or bits per pixel, that it uses. For example, a device with four bits per pixel will support 16 simultaneous colors.

**SMOOTH SCROLLING** A scrolling process by which text characters scroll up or down smoothly one pixel at a time, rather than scrolling one full character line at a time which tends to appear slightly jerky. Smooth scrolled text can easily be read while scrolling is in process. EGA and VGA provide hardware support to assist in smooth scrolling.

**SUPERVGA** Display adapter for PC or PS/2 computer that is compatible with IBM VGA and is capable of displaying 256 simultaneous colors in resolutions of 640 x 400 or above.

**SYNC, SYNC PULSE** Another term for horizontal and vertical retrace pulses to a CRT display.

**TELETYPE MODE** An EGA/VGA BIOS call that displays text as if the display screen were a page in a teletype machine. The cursor is advanced after each character, scrolling and line wrap are performed as needed, carriage return, line feed bell, and backspace characters are recognized.

**TEXT MODE** On EGA and VGA, a display mode in which the display adapter converts ASCII character data into display information directly. Text mode displays impose very little overhead on the system processor, but do not support graphics.

**VERTICAL RETRACE** On CRT displays, the time interval after a raster scan has been completed when the electron beam is returning to the top of the display screen for the next scan. The electron beam is blanked during this time. Retrace occurs between 50 and 70 times a second, depending on the display.

**VGA** The IBM Video Graphics Array display adapter

**VLSI** Very Large Scale Integration - the technology of manufacturing Integrated Circuits (chips) with thousands of transistors on a single device. The personal computer was made possible because of VLSI technology.

**WAIT STATE** When a system processor is reading or writing a memory or peripheral device which cannot respond fast enough, a time interval (usually a fraction of a microsecond) is inserted during which the processor does nothing but wait for the slower device. This has a detrimental effect on system throughput, but is often necessary. Because of the constant requirement to perform screen refresh, many display adapters, including EGA and VGA, impose wait states on the processor.

**WINDOW** As commonly used in personal computers, the term WINDOW refers to a section of the display screen (usually rectangular) that displays data independently of the reset of the screen. Several windows may be present at once on the display.

In advanced computer graphics, the terms WINDOW and VIEWPORT are used to refer to the content and position of display information. The section of data which is to be displayed is referred to as a WINDOW (as if looking at a scene through a window, and only part of the scene is visible.) The position and scaling of the information on the screen is referred to as a VIEWPORT.

# INDEX

## A

## B

## N



## O



## P

## Q



## R

## S

## T

## X



## Y



## Z

# About the Authors

George Sutty of Huntington Beach, California received his BA in mathematics from Vassar College and his MA in mathematics/computer science from UCLA. He is a consultant who now operates Computer Graphics Labs, specializing in the field of computer graphics software and systems. He has developed software for such companies as AST Research, Western Digital (Paradise), Basic Four, and IBM.

Steve Blair of Alta Loma, California is an independent engineering consultant who operates Digital Resources in Alta Loma. He has developed software and circuit designs for companies such as AST Research, Emulex Corporation, General Automation, Computer Automation, and others. He holds a BSEET degree from DeVry Institute of Technology.

# Programming Examples Diskette for Advanced Programmer's Guide to SuperVGAs

This diskette contains source code and make files for programming examples used in the text of the Advanced Programmer's Guide to SuperVGAs. Due to their size, all files had to be compressed to fit onto single diskette. Compression utility PKZIP.EXE has been used to pack the files onto this diskette. To unpack files, utility PKUNZIP.EXE should be used. Both of these utilities are included on the diskette.

Files in the directory DEMOS are demonstration and test programs DEMO.C, SHOW.ASM, and GRAB.ASM, files in directory IMAGES contain sample scanned images PICTUREx.IMG. Files in directories 256COL, 16COL, 4COL, and 2COL, contain drawing routines that are independent of any particular board (one directory for each memory organization type). Rest of the directories contain files with board-dependent and mode-dependent procedures, one directory for each chip manufacturer discussed in the book.

## Quick Start For the Impatient Reader

For each board there are two programs available for 640x480 256-color mode; the program DRAW.EXE to demonstrate drawing routines, and program SHOW.EXE to display scanned images. These must be unpacked from a proper, board dependent, directory. For example for Tseng based boards use the following commands:

```
C:
A:\PKUNZIP A:\TSENG\TSENG DRAW.EXE
A:\PKUNZIP A:\TSENG\TSENG SHOW.EXE
A:\PKUNZIP A:\IMAGES\IMAGES
```

To run the drawing demo program type:

```
DRAW
```

To display scanned images type:

```
SHOW PICTURE0.IMG
```

## How to Transfer Files From the Diskette

Diskette is organized in several directories, and this organization should be preserved so that the make files operate properly. You will need about 2MBytes of space on your disk for all the programming examples. To transfer files onto your hard disk, you may use batch file INSTALL.BAT with a name of the board you are interested in. For example for Tseng based boards you would use the following commands:

```
C:
MD C:\SUPERVGA
CD C:\SUPERVGA
A:\INSTALL TSENG
```

Note that the batch file should be started from the directory where you want the files. If you also need to add to your disk examples for other boards, use utility PKUNZIP.EXE. For example to add ATI examples you would use the following commands:

```
C:
CD C:\SUPERVGA
MD ATI
CD ATI
A:\PKUNZIP A:\ATI\ATI
```

When you get done with transferring the files, the following directories should be in subdirectory C:\SUPERVGA

| **Directories used by all boards:** | |
|---|---|
| 256COL | 256 color drawing routines (packed) |
| 256COLCI | 256 color drawing routines (Cirrus planar) |
| 16COL | 16 color drawing routines (planar) |
| 16COLATI | 16 color drawing routines (ATI packed) |
| 4COL | 4 color drawing routines (Headland style) |
| 4COL01 | 4 color drawing routines (Genoa style) |
| 4COL02 | 4 color drawing routines (Ahead style) |
| 4COLATI | 4 color drawing routines (ATI style) |
| 4COLPACK | 4 color drawing routines (WD style) |
| IMAGES | Images to display with SHOW.EXE |
| DEMOS | Source for DEMO.EXE, GRAB.EXE, SHOW.EXE |

| **Board specific directories:** | |
|---|---|
| TSENG | Tseng specific files |
| ATI | ATI specific files |

If you are pressed for space, you can use PKUNZIP.EXE to unpack only the directories you need. All files in each directory on the diskette are packed into a single *.ZIP file. For example to unpack only 256 color examples for Tseng based boards use the following commands:

```
C:
MD \SUPERVGA
CD SUPERVGA
MD 256COL
CD 256COL
A:\PKUNZIP A:\256COL\256COL
CD ..
MD DEMOS
CD DEMOS
A:\PKUNZIP A:\DEMOS\DEMOS
CD ..
MD TSENG
CD TSENG
A:\PKUNZIP A:\TSENG\TSENG
```

## How to Compile and Link Example Program Demo.exe

Each board dependent directory contains a file named SELECT.ASM containing mode and page selection procedures, files named MODExx.INC containing mode dependent constants, and make file named DEMO used to compile and link demonstration program.

Two command line macros are needed to properly use the make file DEMO. The first one, MODE macro, determines the mode to be used. And the second one, DRAWPATH macro, determines which set of drawing routines to use (4, 16 or 256 color). For example, to build a program DEMO62.EXE for the 640x480 256 color mode on ATI boards use the following commands:

```
CD \ATI
MAKE MODE=62 DRAWPATH=..\256COL DEMO
```

These commands will cause the file MODE62.INC to be copied over MODE.INC (which is used by the SELECT.ASM file), SELECT.ASM will be assembled, drawing routines in directory 256COL will be assembled, test program DEMO.C in directory DEMOS will be compiled, and the resulting object files will be linked to form the DEMO62.EXE demo program.

The make file was prepared with Microsoft C 5.1 and Microsoft Assembler 5.0. To use it with other versions, you may have to prepare you own make file to perform the steps described in the previous paragraph.

## How to Use Programming Examples With Pascal

Due to the size limitations, we were unable to include Pascal versions of the programming examples on this diskette. If you would like to obtain a separate diskette with Pascal examples please include $15.00 for shipping and handling, and write to:

Graphics Software Labs
7906 Moonmist Circle
Huntington Beach, CA 92648

## Format of Scanned Images

Each image file in the directory IMAGES starts with 768 bytes of DAC register values, three bytes (R, G, B) for each index. DAC register values are followed by 480 scan lines of image data, 640 bytes per scan line, with one byte per pixel. Images are courtesy of:

RIX SoftWorks Inc.
1855 MacArthur Blvd
Irvine, California.
(714) 467-8266

These images are not in RIX format.

## How to Find Out More About Compression Utility

PKZIP and PKUNZIP are (c) Copyright of PKWARE Inc. To find out more about the compression utilities and to get documentation contact:

PKWARE Inc.
7545 North Port Washington Road, Suite 205,
Glendale, WI 53217
(414) 352-3670

George Sutty April 3, 1990

# CUSTOMER SUPPORT

It is important that you register your purchase of any Simon & Schuster software package. By completing and returning your Owner Registration Card, you become eligible for:

- Software support directly from S & S.
- Diskette replacement when applicable.
- Purchase of future product upgrades at special prices.
- Subscriptions to Hint Books and newsletters where applicable.

## Software Support

S & S will provide support to registered owners. Our technical support number is (900) 454-8900. It is staffed on working days during normal business hours, 10:00 am to 6:00 pm, Eastern time. There is a charge of $1.00 per minute charged to your phone for each minute after the first.

**Mail-in Support Service**—Registered owners may write to us with questions. We will respond in writing. There is no additional charge for this service.

We realize that our software packages are put to a wide variety of uses, however, we can only answer questions about the software package itself. We cannot support the hardware and operating system required to run our software packages.

## Before Calling Customer Support

Before calling our Technical Support Department, please make sure you have followed the steps in the "Pre-call Checklist" below.

## Pre-call Checklist

1. If you are having difficulty understanding the program, have you read and performed the suggestions listed in the manual?

2. If you are not sure how to operate the program, have you used the help system (where available) to find the answer?

3. If there seems to be a problem in the software, can you reproduce the problem by following your steps again?

4. If the program displayed an error message, please write down the exact message.

5. You should be familiar with the hardware configuration you are using. We may need to know the brand/model of your computer, printer, the total amount of memory available, what video adaptor(s) you have in the system, the operating system version, etc.

6. When you call our Technical Support Department, please be at your computer or be prepared to repeat the sequence of steps leading up to the problem.

## Services and Prices

The above services and prices are subject to change without prior notice.

## Advanced Programmer's Guide to SuperVGAs

Square peg in a round hole?

Get a FREE 3-1/2" disk for Advanced Programmer's Guide to SuperVGAs.

Send in this form and your unused 5-1/4" disk to receive a free 3-1/2" disk for Advanced Programmer's Guide to SuperVGAs.

☐ Yes, I want the Advanced Programmer's Guide to SuperVGAs disk.
ISBN 0-13-040544-2

NAME ______________________________

ADDRESS ______________________________

CITY ______________ STATE ________ ZIP ________

***Please send all requests to:*** Prentice Hall Mail Order Billing, Rte 59 at Brook Hill Drive, West Nyack, NY, 10995-9901. Please allow 4-5 weeks for delivery. For more information call 1(800) 624-0023.

67-04054

## Advanced Programmer's Guide to SuperVGAs
## LIMITED WARRANTY REGISTRATION CARD

*In order to preserve your rights as provided in the limited warranty, this card must be on file with Simon & Schuster within thirty days of purchase.*

**Please fill in the information requested:**

NAME ______________________ PHONE NUMBER (   ) ______________

ADDRESS ______________________________

CITY ______________ STATE ________ ZIP ________

COMPUTER BRAND & MODEL ________ DOS VERSION ________ MEMORY ________ K

**Where did you purchase this product?**

DEALER NAME? ______________________ PHONE NUMBER (   ) ______________

ADDRESS ______________________________

CITY ______________ STATE ________ ZIP ________

PURCHASE DATE ______________ PURCHASE PRICE ______________

**How did you learn about this product? (Check as many as applicable.)**

STORE DISPLAY________ SALESPERSON ________ MAGAZINE ARTICLE ________ ADVERTISEMENT________

OTHER (Please explain) ______________________________

**How long have you owned or used this computer?**

LESS THAN 30 DAYS________ LESS THAN 6 MONTHS ________ 6 MONTHS TO A YEAR ________OVER 1 YEAR ________

**What is your primary use for the computer?**

BUSINESS________ PERSONAL ________ EDUCATION ________ OTHER (Please explain)________

**Where is your computer located?**

HOME________ OFFICE________ SCHOOL ________ OTHER (Please explain) ________

67-01045

## Get the Spark. Get *BradyLine*.

Published quarterly, beginning with the Summer 1990 issue. Free exclusively to our customers.

☐ Check here to begin your subscription.

PUT
FIRST
CLASS
STAMP
HERE

Simon & Schuster, Inc.
Brady Books
15 Columbus Circle
New York, New York 10023

ATTN: **PRODUCT REGISTRATION**

## Programming Examples for Advanced Programmer's Guide to SuperVGAs

**Enclosed is one high-density 1.2MByte diskette for IBM PC, PS/2, and compatibles running DOS 2.0 or higher.**

**On the diskette you'll find several directories divided into three categories:**

**1. Assembly drawing modules that are not dependent on a particular board manufacturer, but that are specific to the number of colors used. There is a directory for 256-color modes (256COL), 16-color modes (16COL), and 4-color modes (4COL).**

**2. Assembly language modules that are specific to particular boards, with mode-dependent *include* and *make* files. There is a separate directory for each manufacturer covered in the text (Ahead, ATI, Chips and Technologies, Cirrus, Genoa, Headland (Video7), Trident/Everex, Tseng/STB, Western Digital (Paradise), Zymos/TrueTech, and VESA/Everex).**

**3. Demo modules in assembly, Pascal, and C. These are board- and mode-independent.**

**All the files are compressed. They can be unpacked using the decompression utility (PKUNZIP.EXE) included on the diskette.**

**For the latest information, read the file README on the Programming Examples diskette.**

Brady Books ■ Distributed by Prentice Hall Trade ■ New York



The disk comes with sample images and a utility to display them on SuperVGAs capable of 640 x 400 or better in 256 colors.

Graphics/IBM

# Exploit the Power of SuperVGA Graphics

*Advanced Programmer's Guide to SuperVGAs* is the only serious reference to the enhanced features and extended display modes of the SuperVGA. It provides in-depth analysis and practical programming tips for virtually all SuperVGA boards on the market today. It also covers the latest standards supplied by the Video Electronics Standards Association (VESA).

All the program examples discussed in the book are included on the companion disk so that you can easily incorporate them into your programs. The disk provides two sets of drawing algorithms. One is supported by all SuperVGA vendors and the other is board-specific.

More than 120 low-level assembly language routines, with some in C and Pascal, give you ready-to-use solutions to hundreds of SuperVGA programming challenges.

Covers all major SuperVGA board and chip manufacturers, including:

■ ATI Technologies (VGA Wonder) ■ Boca Research ■ Headland Technology ■ Orchid ■ Trident ■ STB (VGA Extra) ■ Tseng Labs ■ Hewlett-Packard ■ Cirrus ■ Compaq ■ Zenith ■ Tatung ■ SOTA ■ Chips and Technologies ■ Genoa ■ Sigma Designs ■ Paradise ■ Everex ■ Western Digital ■ Ahead ■ Dell ■ Quadram ■ Video-7 ■ Techmar ■ AST Research

*"This book has more in-depth information on vendor specific implementations than I've seen in one place before."* —Brett Glass
Contributor, *PC Magazine*

Brady Books ■ Distributed by Prentice Hall Trade ■ New York

ISBN 0-13-010455-8

0 21898 10455 7

*Cover illustration by Istvan Banyai*

$44.95